Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Infórmatica

RAMIRO MANUEL SILVA OLIVEIRA

Análise e comparação de métodos *soft/hard* em sistemas reconfiguráveis

Universidade de Aveiro
2010

Departamento de Electrónica, Telecomunicações e Informática

RAMIRO MANUEL SILVA OLIVEIRA

Análise e comparação de métodos soft/hard em sistemas reconfiguráveis

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Valeri Skliarov, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e co-orientação da Dr.ª Iouliia Skliarova, Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho ao meus pais.

o júri

presidente

Prof. Dr. José Alberto Gouveia Fonseca
Professor Associado do Departamento de Electrónica, Telecomunicações e Infórmatica da Universidade de Aveiro

Prof. Dr. Valeri Sklyarov
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Infórmatica da Universidade de Aveiro

Prof. Dr.ª Iouliia Skliarova
Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Dr. Hélio Mendes de Sousa Mendonça
Professor Auxiliar do Departamento de Engenharia Electrónica e de Computadores da Faculdade de Engenharia da Universidade do Porto

agradecimentos

Agradeço aos meus pais, por todo o apoio que me deram, não só durante a realização deste projecto, como também durante a minha vida universitária e não só, agradeço também a minha irmã, e a toda a minha família, que sempre me apoiou.
Agradeço também aos meus amigos, por todo o apoio e amizade, não vale a pena dizer nomes que vocês sabem quem são.
Por fim quero agradecer também ao Prof. Valeri Sklyarov e a Prof. Iouliia Skliarova pela paciência e por toda a ajuda dada durante este projecto.

Obrigado por tudo.

Ramiro

palavras-chave

Field-Programmable Gate Array, PowerPC, Processadores Embutidos, Máquinas de Estados Finitos

resumo

O avançar da tecnologia na área dos sistemas digitais, que se traduziu num aumento significativo da velocidade e numa diminuição dos consumos e das dimensões, levou ao aparecimento do conceito de SoC (System on Chip). Como o próprio nome indica, estes sistemas são compostos por diversos componentes, todos na mesma pastilha de silício (microprocessadores, memórias, ADC, controladores I/O, etc.).
Nesta dissertação pretende-se comparar alguns desses sistemas e analisar as suas vantagens e desvantagens, quando comparados com outras soluções existentes.
Ao longo deste trabalho são feitas comparações de desempenho entre um processador embutido numa FPGA, uma FPGA e um PC. Para tal são usados algoritmos com base em máquinas de estados finitos.

keywords

Field-Programmable Gate Array, PowerPC, Embedded Processors, Finite-State Machines
.

abstract

The advance of the technology in the area of digital systems, which as translated itself in a significant increase of the speed and in a decrease in consumptions and dimensions, has led to the appearing of the SoC concept (System on Chip). As the name indicates, these systems are composed by several components, all in the same silicon wafer (microprocessors, memories, ADC, I/O controllers, etc.).
In this dissertation it is intended to compare some of those systems, and analyze their advantages and disadvantages, when compared with other existing solutions.
Throughout this work, comparisons between an embedded processor, a FPGA and a PC are made. In order to do this Finite-State Machine based algorithms are used.
.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1 - *Introdução*

1.1 Enquadramento

O avançar da tecnologia na área dos sistemas digitais, que se traduziu num aumento significativo da velocidade e numa diminuição dos consumos e das dimensões, levou ao aparecimento do conceito de *SoC* (*System on Chip*). Como o próprio nome indica, estes sistemas são compostos por diversos componentes, todos na mesma pastilha de silício (microprocessadores, memórias, ADC, controladores I/O, etc.). Estes sistemas tornaram a construção de sistemas electrónicos mais barata, sendo também mais eficientes e com menor consumo, o que levou a que os projectistas aderissem a utilização dos mesmos.

Dentro destes dispositivos podemos encontrar a FPGA. Desde a sua introdução em 1985 pela *Xilinx* [7], que esta tem sofrido um grande desenvolvimento, quer na sua capacidade lógica, quer na sua estrutura, visto hoje em dia incorporar uma grande variedade de componentes digitais embutidos, tais como processadores e outros.

Actualmente podem ser encontradas variadas FPGA com processadores embutidos, que além de facilitarem a tarefa de programação, visto usarem linguagens de programação de alto nível e amplamente conhecidas, como C/C++, ao contrário das linguagens de descrição de *hardware*, permitem também um acesso directo ao processador. Estes processadores podem ir de pequenos microcontroladores, com capacidade para executar poucos milhões de operações por segundo, até sistemas paralelos compostos por vários processadores de elevado desempenho.

1.2 Objectivos e Motivação

Com o aumentar da capacidade das FPGA, tornou-se possível incorporar nestas, processadores com uma certa complexidade. Tais processadores podem estar directamente inseridos na pastilha de silício (*Hard-core*), ou podem ser implementados através de um processo

de síntese (*Soft-core*). Como estes dispositivos têm uma grande flexibilidade torna-se possível acoplar a estes periféricos. Os periféricos ligados ao processador provêm de componentes existentes em bibliotecas IP, ou podem ser construídos pelo projectista. Depois de terminada a construção, é possível programá-lo em C/C++.

Como tal, é interessante verificar se a utilização de um processador embutido é, não apenas mais fácil, mas também mais eficiente do que uma FPGA.

1.3 Metodologia

O objectivo deste trabalho é verificar que método é mais eficaz para correr certos algoritmos, neste caso algoritmos baseados em máquinas de estados finitos recursivas hierárquicas. O plano proposto inicialmente suponha correr vários algoritmos diferentes, no entanto foi apenas possível correr um algoritmo.

Os vários testes foram feitos usando o *Xilinx* ISE para correr os algoritmos na FPGA e o *Xilinx* EDK para correr os algoritmos no processador embutido, assim como o *Microsoft Visual Studio 2010* para correr o algoritmo no PC.

1.4 Estrutura da dissertação

Esta tese encontra-se dividida em 5 capítulos. Para além deste, existem ainda os seguintes:

- Capitulo 2 – Neste capítulo é feita uma introdução a vários conceitos, como sistemas computacionais e sua história e evolução, entre outros
- Capitulo 3 – Ao longo deste capítulo são referidas e identificadas as ferramentas usadas neste trabalho, os microcontroladores, assim como o *software* usado ao longo deste trabalho.
- Capitulo 4 – No quarto capítulo é feita uma descrição do trabalho efectuado nas várias plataformas, assim como também certos conceitos que ajudaram a fazer a comparação entre as várias plataformas
- Capitulo 5 – No último capítulo são apresentados os resultados obtidos, assim como também são mostradas as conclusões e enumeradas algumas possibilidades para trabalhos futuros.
- Anexo A – *Xilinx Board Description* FX12
- Anexo B - Algoritmo usado no PC e em PowerPC

Capítulo 2 - *Sistemas Digitais Reconfiguráveis*

2.1 Introdução

A grande evolução dos dispositivos lógicos programáveis, normalmente referidos como PLD, permite neste momento projectar sistemas digitais de grande complexidade, recorrendo a ferramentas que se encontram acessíveis a todos. Neste capítulo é feita uma introdução a variados conceitos, entre estes os sistemas computacionais, sistemas embutidos com base em FPGA, etc., sendo também feita uma comparação entre os sistemas tradicionais e os sistemas embutidos.

2.2 Sistemas computacionais

Por norma os sistemas computacionais são divididos em dois grupos, relativamente à sua flexibilidade e aplicação. De um lado podemos encontrar o computador pessoal (PC), extremamente flexível, podendo executar variadas aplicações, recorrendo a um sistema operativo que gere os recursos do sistema. Do outro lado podemos encontrar os sistemas computacionais especializados, nomeadamente os sistemas embutidos, que por norma correm uma aplicação em permanência. Todavia estas diferenças, estes dois sistemas partilham princípios básicos de operação assim como uma arquitectura física semelhante, o *hardware*. A sua principal diferença é portanto a aplicação a que se destinam, diferindo portanto ao nível do *software*. Como tal, podemos considerar um sistema computacional, ao nível do *hardware*, como, um processador que executa aplicações, memória de variados tipos para armazenar tanto programas, como dados, e finalmente por dispositivos de entrada e saída (I/O) que têm como função a comunicação com o exterior, rato, teclado, monitor, etc., conforme ilustrado na figura 2.1.

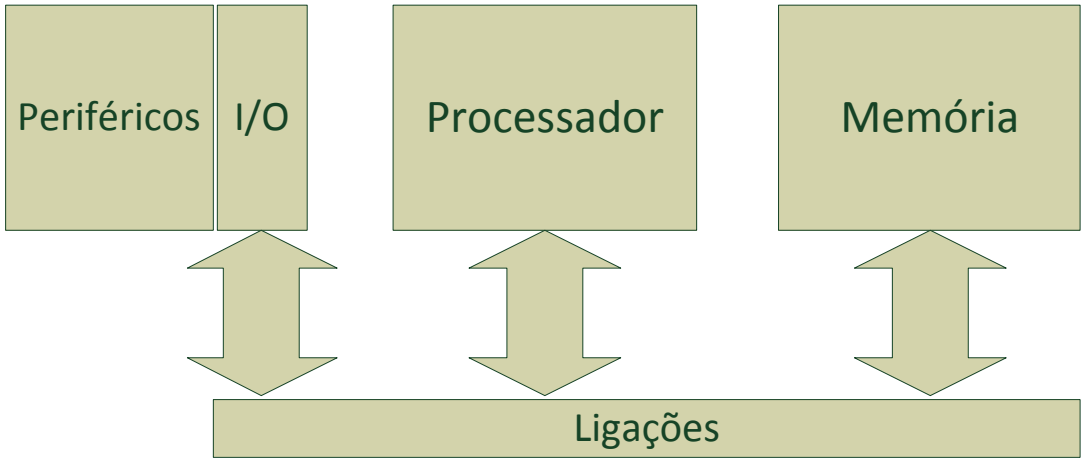

Figura 2.1 - Sistema Computacional básico

Neste sistema o mais importante é o processador, que manipula e processa a informação seguindo uma determinada ordem, formando assim um programa. Um tipo de processador muito utilizado é o microprocessador, muitas vezes chamado CPU. Este tipo de processador está implementado num único circuito integrado, sendo talvez os mais famosos os Pentium da Intel, existindo no entanto outros. A implementação dentro do mesmo circuito integrado de um processador, memória e alguns dispositivos de I/O é normalmente designado por microcontrolador. A comunicação entre estes é feita dentro do próprio circuito integrado, esta solução tem por norma como finalidade o uso em sistemas embutidos. Existe uma alargada variedade de microcontroladores disponíveis, desde pequenas PICs e AVRs até processadores *PowerPC* com controladores I/O embutidos.

As últimas quatro décadas tem sido pródigas na evolução tecnológica relativa aos circuitos integrados, obedecendo a Lei de Moore, que diz que num período de 18 a 24 meses o número de transístores presentes num circuito integrado duplica [9]. Analisando a figura 2.2, podemos observar esse crescimento.

Microprocessor	Year of Introduction	Transistors
4004	1971	2,300
8008	1972	2,500
8080	1974	4,500
8086	1978	29,000
Intel286	1982	134,000
Intel386™ processor	1985	275,000
Intel486™ processor	1989	1,200,000
Intel® Pentium® processor	1993	3,100,000
Intel® Pentium® II processor	1997	7,500,000
Intel® Pentium® III processor	1999	9,500,000
Intel® Pentium® 4 processor	2000	42,000,000
Intel® Itanium® processor	2001	25,000,000
Intel® Itanium® 2 processor	2003	220,000,000
Intel® Itanium® 2 processor (9MB cache)	2004	592,000,000

Figura 2.2 - Lei de Moore [9]

2.3 Sistemas embutidos

Por norma, os sistemas embutidos têm como finalidade sistemas com exigências de baixo custo e consumo de energia, poucos componentes, resposta em tempo real e uma coexistência entre *hardware* e *software*. Apesar do utilizador comum ter menos noção da sua existência, estes tem uma quota de mercado muito maior, podendo ser encontrados em variados tipos de sistemas usados no dia-a-dia em casas, (TVs, DVDs, consolas de jogos), em escritórios (impressoras, WLAN), na indústria automóvel (controlo do motor, travões ABS), na indústria aeronáutica (navegação, piloto-automático), em dispositivos de uso pessoal (telemóveis, leitores de mp3, PDAs, cartões electrónicos). Por norma os sistemas embutidos costumam ser mais simples e com menor capacidade de processamento que um PC, mas algumas aplicações requerem um poder de processamento maior, tal como um sistema distribuído de controlo de um avião. Em geral, um sistema embutido não possui sistema operativo e quase nunca permite a instalação de um *software*, estando o seu *software* presente numa memória permanente. Em variadas situações estes sistemas são usados para substituir circuitos electrónicos específicos, pois o uso de microprocessadores embutidos leva a que a resolução do problema seja feita por *software* e não por *hardware*, o que em aplicações mais complexas simplifica a sua construção e manutenção.

Com a diminuição do tamanho do transístor e com o crescimento da densidade lógica, os sistemas digitais registaram um grande crescimento, como um aumento das suas competências e

velocidades, sofrendo assim um aumento também na complexidade. Assim, este avanço da tecnologia deu origem a um novo conceito de sistema integrado, o *SoC*, onde existe uma implementação de sistemas embutidos integrados num só circuito. Com isto tornou-se possível criar sistemas onde todos os componentes (processador, memória, e controladores de comunicação I/O) estão presentes dentro de mesmo circuito integrado. Esta abordagem apresenta inúmeras vantagens provenientes da interacção muito próxima entre os diferentes componentes do sistema, pois permite um aumento da velocidade, e uma diminuição do tamanho, custo e consumo de energia do sistema, correspondendo deste modo às exigências de variadas aplicações. Como tal este tipo de sistemas está cada vez mais presente em aplicações à nossa volta. Deste modo os sistemas embutidos são os sistemas com o maior crescimento na indústria dos sistemas computacionais, uma vez que podem ser usados em um número crescente de aplicações.

2.4 Sistemas de Controlo Embutidos

Esta evolução dos circuitos integrados nas últimas quatro décadas revolucionou os sistemas electrónicos, pois estes tinham como base componentes como, amplificadores operacionais, resistências, condensadores e bobinas. Estes sistemas tinham como vantagem oferecerem processos paralelos, mas tinham como grande desvantagem a pouca fiabilidade, devido à temperatura e ao envelhecimento. O microprocessador 4004 da Intel foi a primeira plataforma digital configurável através de *software*. A partir daí com o aparecimento de componentes digitais cada vez mais diversificados, o controlo embutido passou em grande maioria para o digital. Nos sistemas de controlo embutido podemos identificar variadas plataformas digitais, cada uma com as suas vantagens e desvantagens. Dentro deste meio as plataformas que podem ser usadas actualmente para implementação de um controlador digital são:

- Microprocessadores
- Microcontroladores
- ASSPs *(Application Specific Standard Product)*
- FPGAs

Como tal, quando pretendemos implementar um qualquer controlador digital, temos que ter em conta todas a características inerentes a cada uma destas soluções. Assim, conforme a aplicação final, as necessidades que levam a escolha de uma plataforma, podem ser as seguintes:

- Consumo energético

- Preço
- Tempo de resposta em tempo real
- Quantidade de produção
- Ferramentas de programação
- Bibliotecas e código portável e utilizável

2.5 Microprocessador

Como já foi referido o microprocessador foi a primeira plataforma digital configurável por *software*, revolucionando a metodologia dos sistemas digitais como nenhum outro componente, continuando ainda a ser a escolha de referência em variadas áreas. Uma destas áreas é a área de aplicações em tempo real, com elevadas taxas de actualização, que requerem que o processador seja programado no seu *assembly* nativo. Apesar de a grande maioria dos microprocessadores comerciais usados hoje sejam utilizados em aplicações de dados, há microprocessadores embutidos em microcontroladores para aplicações de controlo em tempo real. Estes usam interrupções para diferentes objectivos dentro do mesmo controlador. Cada interrupção é gerada em função do tempo de execução de determinada tarefa [21].

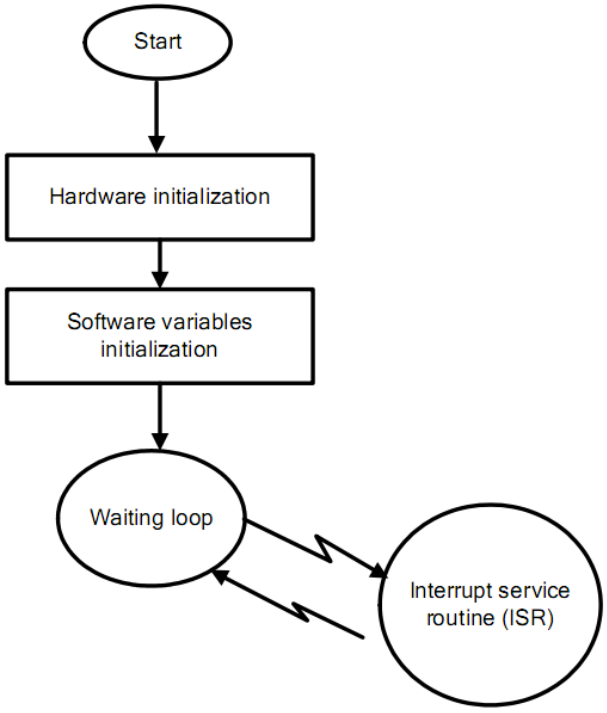

Figura 2.3 - Rotina de Serviço as Interrupções [21]

Na figura 2.3 podemos observar um esquema do funcionamento de um sistema de controlo digital através de rotinas de serviço de interrupção. Como podemos observar, sempre que é activada uma interrupção, o microprocessador executa o respectivo algoritmo e actualiza os

seus resultados. A grande desvantagem deste tipo de solução é que o microprocessador (de uso geral) pode apenas executar uma instrução de cada vez, e por isso pode não ser suficientemente rápido para atender múltiplas interrupções de diferentes aplicações de um controlador, sendo assim incapaz de implementar controladores mais complexos

2.6 Microcontrolador

Um microcontrolador, por norma, combina um processador com vários periféricos no mesmo circuito integrado, fornecendo assim mais soluções para controladores em sistemas embutidos, pois o facto de combinarem periféricos, que antes eram externos no circuito integrado, permite criar sistemas mais complexos, oferecendo assim maior integração e consequentemente um menor custo e tamanho mais reduzido que os microprocessadores.

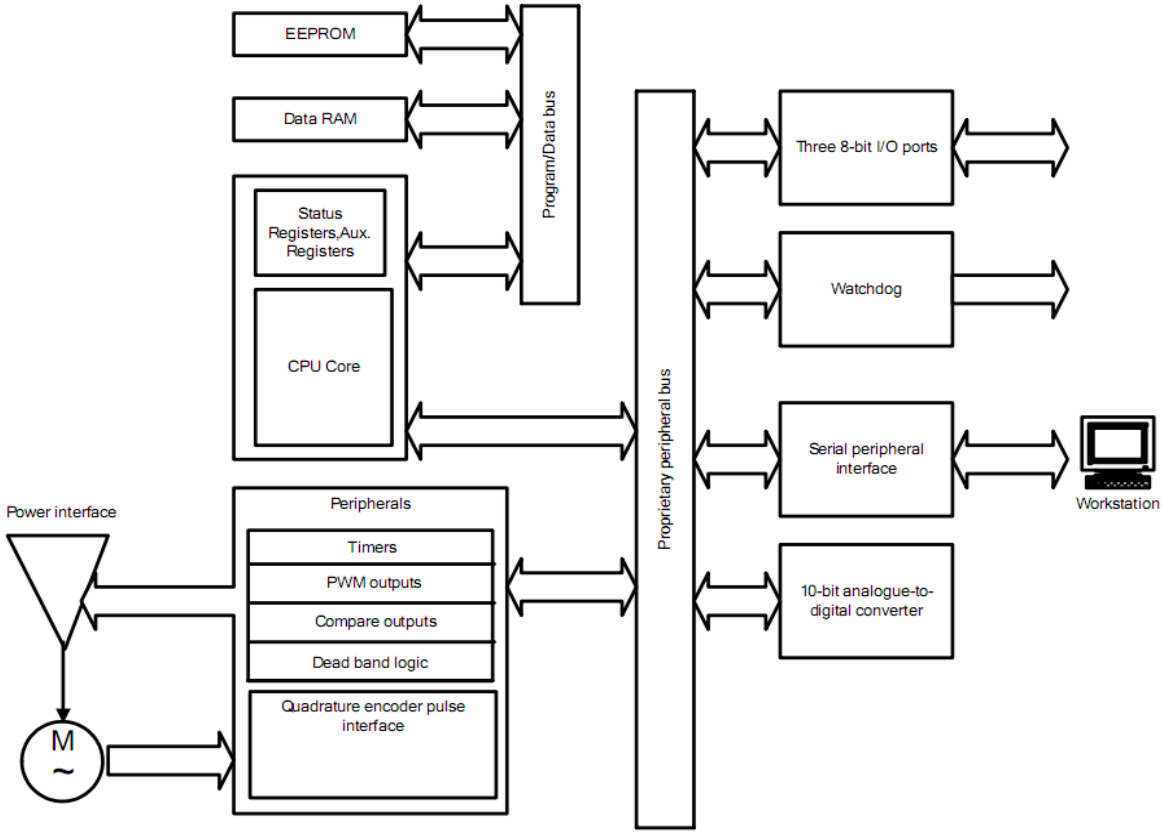

Figura 2.4 - Exemplo de um microcontrolador usado no controlo de um motor [21]

Do mesmo modo que nos microprocessadores, a divisão das tarefas vai ser feita de acordo com a de necessidade uma taxa de actualização maior ou menor. Assim, se a taxa de actualização necessária for pequena podemos usar linguagens de programação, no entanto se a taxa de actualização necessária for elevada, vamos ter que recorrer ao *assembly* nativo do

respectivo microcontrolador. Deste modo, são usados compiladores e linguagens *assembly* para converter programas escritos em linguagens de alto-nível, em código máquina, para em seguida armazenar esse código na memória embutida do microcontrolador.

Um dos exemplos mais comuns de utilização de um microcontrolador é o controlo de um motor. Para a maioria das aplicações o microcontrolador é a solução mais adequada, embora este tipo de solução possa ter como grande desvantagem possuir um número fixo de periféricos. Embora existam no mercado uma grande variedade de microcontroladores, nem sempre é possível encontrar um que cumpra com exactidão todas as exigências de uma determinada aplicação [21].

2.7 ASSPs

Uma ASSP é, como o próprio nome indica, um componente lógico configurado para uma aplicação específica, podendo ser aplicado numa extensa gama de sistemas, tendo assim uma grande procura devido as suas características. Estes dispositivos são usados em todas as indústrias, deste as comunicações até à automóvel. Por norma as ASSP são controladas através de uma palavra de controlo, todavia algumas, mais configuráveis, fornecem endereços, dados e controlo de barramentos para serem ligados a um processador de um sistema que os controla [21].

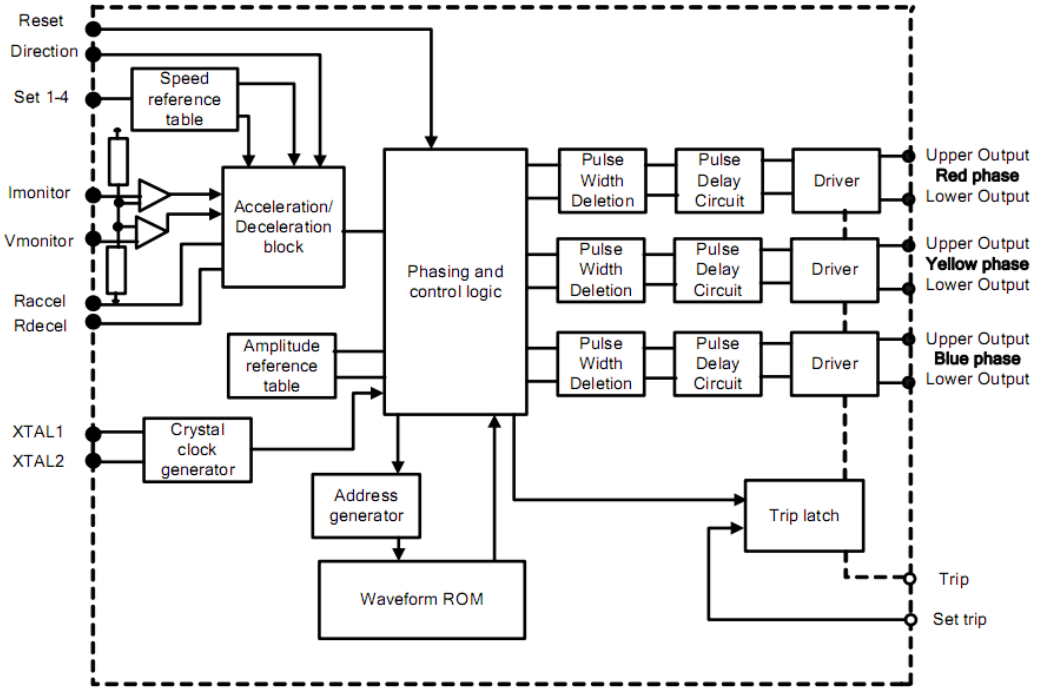

Figura 2.5 - Chip ASSP para o controlo de um motor de indução AC de 3 fases [21]

2.8 FPGA

2.8.1 Introdução

Para o fim ficaram as FPGAs, a base deste trabalho. Numa FPGA há a possibilidade de num mesmo circuito integrado colocar um processador e toda a lógica necessária e específica a uma dada plataforma. Assim, uma FPGA junta as vantagens das três plataformas enunciadas anteriormente. Todavia, os microcontroladores são muitas vezes escolhidos em detrimento das FPGAs devido ao preço inferior e ao menor consumo de energia, no entanto, o mercado das FPGAs tem vindo a crescer devido as variadas vantagens oferecidas pelas mesmas, tais como a possibilidade de reutilizar código e bibliotecas, a portabilidade do código para vários fornecedores e ainda ferramentas de programação de baixo custo. Em adição a estas vantagens, existem imensos núcleos de propriedade intelectual que permitem aumentar muito a produtividade, pois possibilitam uma prototipagem rápida dos sistemas. Enquanto anteriormente estes dispositivos tinham por norma pouca capacidade, hoje em dia são capazes de suportar sistemas lógicos de grande capacidade, sendo assim possível, usando apenas uma FPGA, criar sistemas embutidos com controladores de elevada complexidade. Deste modo o preço cada vez mais acessível e a crescente capacidade, aliadas à reconfigurabilidade, permitem uma grande flexibilidade no desenho de sistemas embutidos digitais. Assim, os sistemas baseados em FPGAs são cada vez mais economicamente viáveis para variadas aplicações, visto integrarem num único circuito integrado variados componentes.

Figura 2.6 - FPGA [11]

2.8.2 Evolução das FPGA

Durante a década de 80 grande parte dos sistemas digitais eram criados integrando vários componentes já existentes, como microprocessadores, controladores I/O, etc.. Logo quando se pretendia criar um sistema mais complexo, era necessário um elevado número de componentes, o que se traduzia num sistema de grandes dimensões, com uma frequência máxima reduzida. Isto levou ao aparecimento de circuitos integrados personalizados, que num único chip incorporavam todos os componentes necessários para um projecto, no entanto, devido a serem personalizados, a sua construção era dispendiosa e morosa, tudo isto tornava a sua produção em pequenas quantidades pouco atractiva.

A solução foram as FPGA, que apareceram no mercado em 1985 pela mão da *Xilinx*, permitindo o desenvolvimento de sistemas digitais, de grande complexidade, em um único chip, com um tempo de projecto muito curto devido às potentes ferramentas CAD.

Observando a figura 2.7 podemos concluir que, desde a introdução da FPGA no mercado, houve uma grande evolução na sua arquitectura sendo que, hoje em dia podemos encontrar implementados dentro do mesmo chip processadores usando bibliotecas IP, programados mais tarde através de linguagens como C e C++, oferecendo uma grande flexibilidade do desenvolvimento de aplicações.

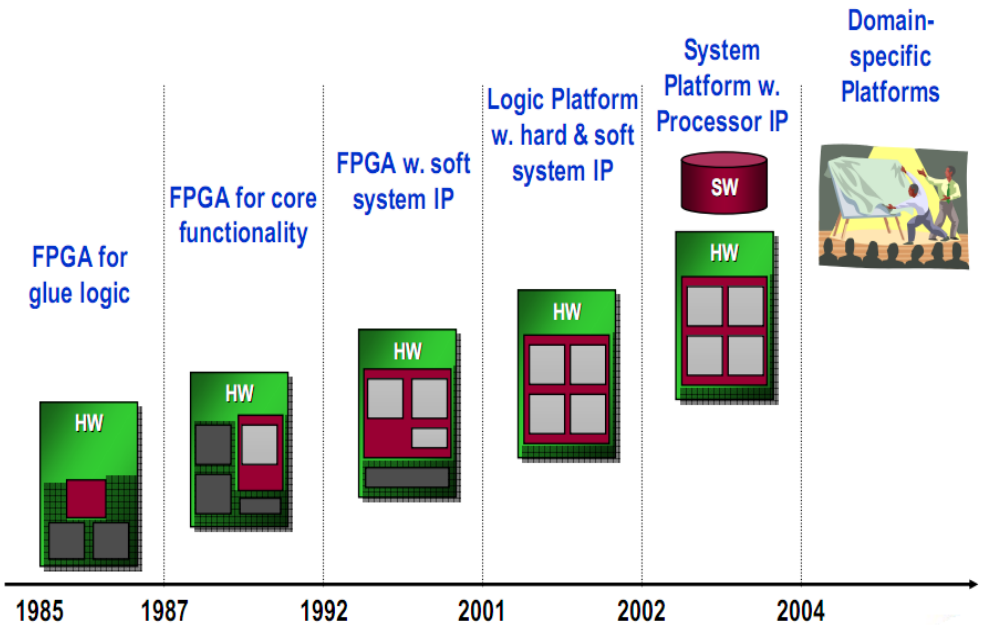

Figura 2.7 - Evolução da arquitectura das FPGA [12]

2.8.3 Estrutura Interna

A estrutura interna de uma FPGA é constituída por vários tipos de componentes, entre eles, IOBs, CLBs entre outros, como pode ser visto na figura 2.8.

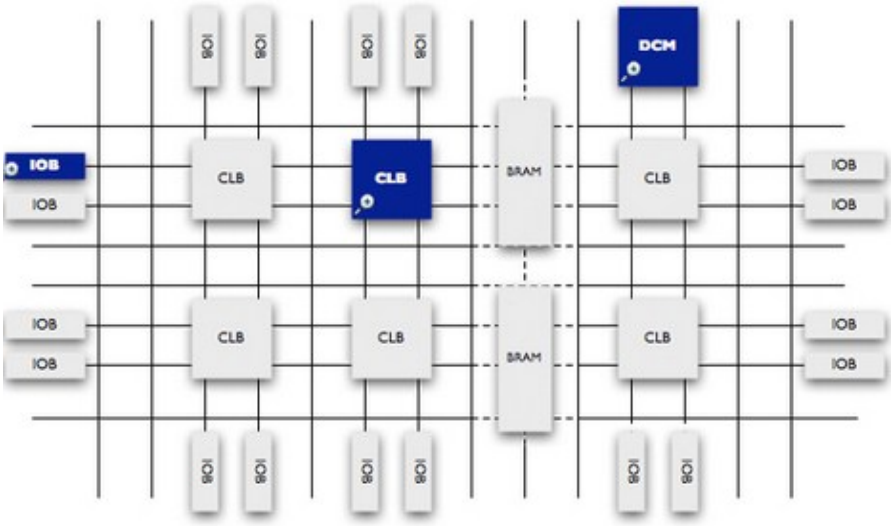

Figura 2.8 - Estrutura Interna FPGA [14]

2.8.3.1 IOB (Input/Output Block)

Actualmente as FPGAs oferecem suporte a variados *standards* de I/O, fornecendo assim imensas possibilidades de interacção com diversos componentes. Hoje em dia uma FPGA oferece mais de uma dúzia de bancos de I/O, permitindo assim uma grande flexibilidade no suporte de I/O [14].

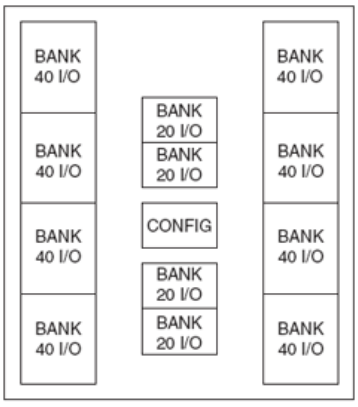

Figura 2.9 - IOB [14]

2.8.3.2 **CLB (Configurable Logic Block)**

É a unidade básica numa FPGA. O seu número e as suas características variam de FPGA para FPGA, mas cada CLB consiste numa matriz de troca configurável com 4 ou 6 entradas, alguns circuitos de selecção e flip-flops. A matriz de troca é altamente flexível e pode ser configurada para lidar com lógica combinatória, RAM ou *shift-registers* [14].

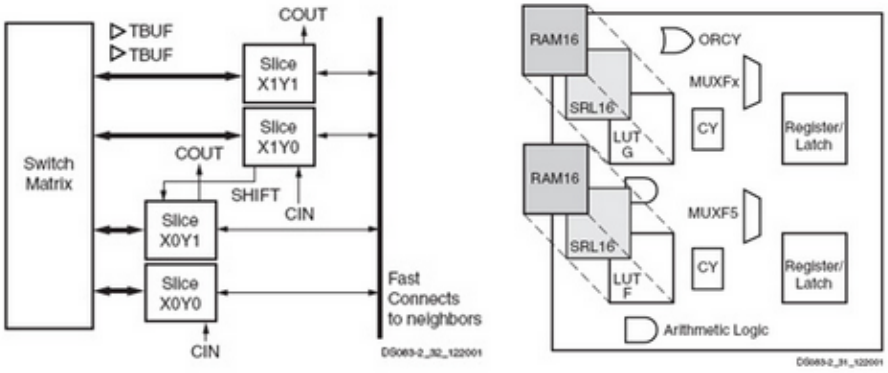

Figura 2.10 – CLB [14]

2.8.3.3 **DCM (Digital Clock Management)**

Esta característica é fornecida pela maioria das placas no mercado, e praticamente eliminou o *skew* (i.e. sinal do relógio não chega síncrono a todos os componentes), variações de temperatura, acoplamentos capacitivos e outros problemas que os projectistas tinham que enfrentar quando criavam sinais globais em FPGA no passado [14].

2.8.3.4 **Ligações**

Enquanto o CLB fornece a capacidade lógica, ligações flexíveis encaminham os sinais entre CLBs e de e para os blocos de I/O. As ligações podem ser de vários tipos, das desenhadas para ligar CLBs, a ligações com baixo *skew* para sinais de relógio e outros sinais. Os *software*s tornam as ligações invisíveis ao utilizador, excepto quando especificado pelo utilizador, o que reduz significativamente a complexidade do projecto [14].

2.8.3.5 **Memória**

Blocos de memória embutidos existem em quase todas as FPGA, o que permite um projecto com memória on-chip. As FPGA fornecem até 10Mb de memória on-chip. Hoje em dia a maior parte das FPGA possui incorporadas memórias RAM, normalmente chamadas Block RAM. O

seu número varia de dispositivo para dispositivo. Esta memória permite a diminuição dos blocos CLB utilizados, facilitando a criação de sistemas digitais. [14]

2.8.4 Usos das FPGA

O mercado das FPGA está em grande expansão. Devido à sua grande versatilidade é cada vez mais usada, permitindo, com um custo relativamente baixo, construir plataformas e sistemas reconfiguráveis. As FPGA são usadas nas seguintes áreas, entre outras:

- Aeroespacial/Militar
- Automóvel
- Emissões Áudio e Visuais
- Doméstico
- Industrial, Científica e Médica
- Comunicações com e sem Fios

2.8.5 Fabricantes

Os principais fabricantes são a *Xilinx* e a *Altera*, juntos controlam cerca de 80% do mercado, com a *Xilinx* a controlar cerca de 50%. Existem outros fabricantes como a *Lattice Semiconductor, Actel, QuickLogic*, etc.. Como a FPGA usada neste trabalho é da *Xilinx*, será feita em seguida uma abordagem aos seus produtos.

2.8.6 *Xilinx*

A *Xilinx* foi a inventora da FPGA, sendo hoje em dia a principal fabricante das mesmas. Fundada em 1984, possui hoje em dia mais de 3000 empregados, tendo até hoje registado mais de 2000 patentes. Foi também a primeira companhia a praticar o *outsourcing* do fabrico de componentes. O seu alcance expande-se por variadas áreas e fabrica diversos produtos. Até 2010, a *Xilinx* fornecia dois principais grupos de famílias de FPGAs, as *Spartan* de baixo custo, e as *Virtex* topo de gama. A partir de 2010, vai passar a oferecer três, as *Virtex,* que continuarão a ser topo de gama, introduzindo também duas novas famílias, as *Kintex*, que se posicionarão no meio do mercado, e as *Artix,* que serão de baixo custo. Todas estas famílias terão transístores de 28nm [27]. Podemos na figura 2.11 a comparação entre famílias de FPGA de gerações anteriores.

FPGA Comparison Table				
Features	**Virtex-6**	**Virtex-5**	**Spartan-6**	**Extended Spartan-3A**
Logic Cells	Up to 760,000	Up to 330,000	Up to 150,000	Up to 53,000
User I/Os	Up to 1200	Up to 1200	Up to 570	Up to 519 I/O
I/O Standards Supported	Over 40	Over 40	Over 40	Over 20
Clock Management Technology	PLL	DCM + PLL	DCM + PLL	DCM
Embedded Block RAM	Up to 38 Mbits	Up to 18 Mbits	Up to 4.8 Mbits	Up to 1.8 Mbits
Embedded Multipliers for DSP	Yes (25 x 18 MAC)	Yes (25 x 18 MAC)	Yes (18 x 18 MAC)	Yes (18 x 18 MAC)
Multi-Gigabit High Speed Serial	6.5 Gbps, beyond 11.18 Gbps	3.75 Gbps, 6.5 Gbps	3.125 Gbps	No
PCI Express Technology	Gen 1, x8, hard Gen 2, x8, hard	Gen 1, x8, hard Gen 2, x8, soft	Gen 1, x1, hard	No
Soft Processor Support	Yes	Yes	Yes	Yes

Figura 2.11 - Comparação entre famílias de FPGAs [14]

2.8.7 Processadores embutidos em FPGA

Como já foi referido, hoje em dia a grande parte dos fabricantes de FPGAs oferecem processadores embutidos fisicamente em silício (*hard* IP), ou processadores que podem ser implementados através de lógica programável (*soft* IP). Deste modo, podemos ter um processador com blocos de lógica convencionais, tudo dentro do mesmo circuito integrado, combinando *software* e *hardware*, disponibilizando flexibilidade total no controlo de uma dada aplicação.

Existem diversos algoritmos difíceis de implementar em HDL, que não sendo críticos em relação ao tempo de resposta, podem ser implementados através de um processador embutido na FPGA (*software*), enquanto outras tarefas de controlo, mais sensíveis relativamente ao tempo de resposta são implementadas em *hardware*, usando os recursos lógicos programáveis da FPGA. De modo a combinar *hardware* e *software*, os fornecedores de FPGA disponibilizam um conjunto de ferramentas que facilitam este tipo de implementações.

Processor name	Type/Bits	Interface bus	FPGA vendor
MicroBlaze™	Soft/32	IBM Coreconnect	Xilinx
NIOS®	Soft/32	Avalon	Altera
LatticeMico32	Soft/32	Wishbone	Lattice
CoreMP7	Soft/32	APB	Actel
ARM Cortex-M1	Soft/32	AHB	Vendor independent
LatticeMico8	Soft/8	Input/Output ports	Lattice
Core8051	Soft/8	Nil	Actel
Core8051s	Soft/8	APB	Actel
PicoBlaze™	Soft/8	Input/Output ports	Xilinx
PowerPC	Hard/32	IBM Coreconnect	Xilinx
AVR	Hard/8	Input/Output ports	Atmel

Figura 2.12 – Processadores embutidos [21]

Podemos ver aqui na figura 2.12 alguns processadores existentes no mercado, tanto implementados em *hardware* como em *software*.

2.9 Linguagens de descrição de *Hardware* (HDL)

Em electrónica, uma HDL é qualquer linguagem de uma classe de linguagens de computador usadas para descrição formal de circuitos electrónicos. Pode ser usada para descrever o método de operação do circuito, o seu desenho e testar o seu funcionamento simulando-o. Em contraste com uma linguagem de programação de software, a sintaxe e a semântica de uma HDL inclui notações explícitas para expressar tempo e simultaneidade [20]

As principais linguagens são VHDL e *Verilog*, no entanto hoje em dia é possível programar com linguagens de alto nível como *Matlab* e C++, que posteriormente são convertidas em linguagens de descrição de *hardware* [19].

2.10 Conclusões

Neste capítulo foi feita uma abordagem aos sistemas computacionais e a vários tipos de sistemas englobados dentro dessa mesma descrição. A abordagem as FPGA foi mais aprofundada devido ao facto de serem a base deste trabalho.

Capítulo 3 - *Ferramentas de apoio*

Neste capítulo vão ser discutidos e apresentados os sistemas usados no decorrer desta tese, nomeadamente a placa Nu Horizons FX12 e o *PowerPC* 405. Serão também introduzidas algumas ferramentas tais como o *Xilinx* ISE, o *Xilinx* EDK e o Microsoft Visual Studio 2010 usado como método de comparação. Para além disso, serão discutidos alguns detalhes do fluxo de projecto das respectivas ferramentas. No entanto, vai ser dado um especial destaque ao processador *PowerPC*405.

3.1 Introdução

Como já foi referido anteriormente, os microcontroladores são uma solução que integra um sistema computacional, num único dispositivo físico, sendo que no mesmo dispositivo podemos encontrar, o processador, a memória de instruções e dados, portos de entrada e saída e ainda alguns periféricos, naturalmente todos estes dispositivos estão ligados internamente. Na prática podemos dizer que a utilização de um microcontrolador apenas requer a correcta interligação dos portos de entrada e saída, à aplicação desejada, a respectiva programação e a alimentação.

A evolução das FPGAs veio revolucionar a criação de sistemas integrados complexos num único dispositivo, e é actualmente reconhecida como sendo a melhor solução para os requisitos do mercado. Anteriormente foi dito que o desenvolvimento de um sistema integrado passa pela concepção de todo o *hardware* e *software* necessários à aplicação alvo, neste caso a *Xilinx* disponibiliza a ferramenta EDK, que permite a construção completa de um sistema embutido de controlo especializado. Esta ferramenta pode ter o processador *PowerPC* como unidade de processamento e permite adicionar núcleos de propriedade intelectual pré-concebidos ou criados, para aumentar as funcionalidades do sistema, sendo isto feito através da ferramenta de síntese ISE também disponibilizada pela *Xilinx.*

3.2 *PowerPC*

3.2.1 Introdução

O *PowerPC*405 é uma implementação de 32-bit da arquitectura *PowerPC* para sistemas embutidos, que é derivada da arquitectura *PowerPC*. Esta arquitectura define parâmetros que garantem a implementação de processadores compatíveis ao nível das aplicações, permitindo assim uma flexibilidade alargada no desenvolvimento de implementações derivadas desta arquitectura, que vão ao encontro das necessidades específicas do mercado.

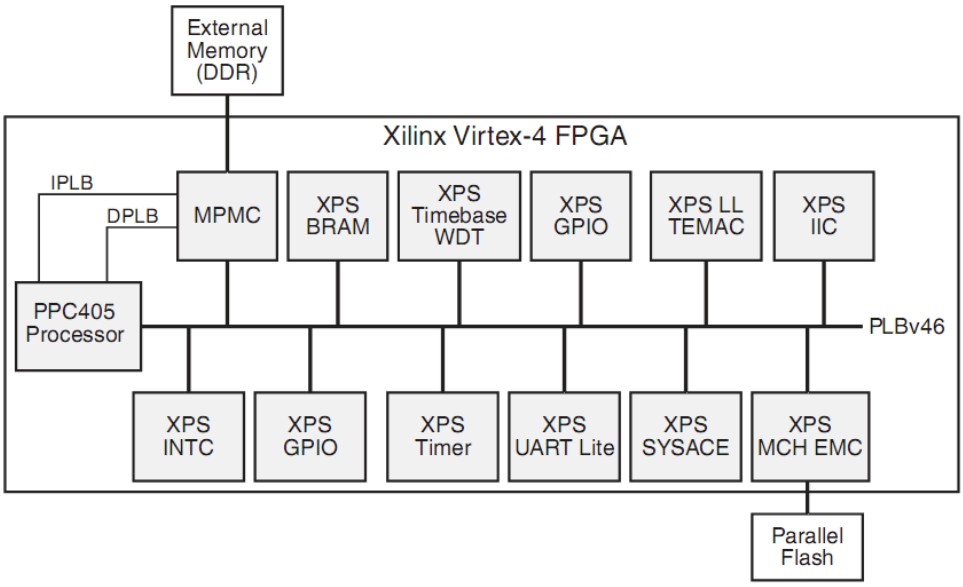

Figura 3.1 - Arquitectura *PowerPC* da Virtex-4 [13]

3.2.2 Detalhes da Arquitectura *PowerPC*

Esta arquitectura é de 64-bit com um *subset* de 32bit, sendo que vão ser apenas enunciados os aspectos da arquitectura de 32 bits referente ao PPC405.

Por norma a arquitectura PPC define o seguinte:

- Conjunto de Instruções
- Modelo de Programação
- Modelo da memória
- Modelo de excepções
- Modelo de gestão da memória
- Modelo de gestão temporal

3.2.2.1 Conjunto de instruções

Especifica o tipo de instruções (como *load/store*, aritmética de inteiros, etc.), as instruções específicas, e a codificação usada para as instruções. A definição do conjunto de instruções especifica também os modelos de endereçamento usado para aceder à memória.

3.2.2.2 Modelo de Programação

Define o conjunto de registos e convenções de memória, incluindo detalhes relacionados com ordenação de bit e byte, e convenções de como os dados são armazenados.

3.2.2.3 Modelo da Memória

Define o tamanho do espaço para endereços e a sua subdivisão em páginas, define também atributos para especificar *memory-region cacheability*, ordenação de byte, (*big-endian* ou *little-endian*), coerência e protecção.

3.2.2.4 Modelo de excepção

Define o conjunto de excepções e as condições que podem causar excepções. Este modelo especifica características de excepção, tais como se são precisas ou imprecisas, síncronas ou assíncronas e *maskable* ou *non-maskable*. Define ainda os vectores de excepção e o conjunto de registos usado quando ocorre uma interrupção como resultado de uma excepção, o modelo garante ainda espaço na memória para excepções específicas de uma aplicação.

3.2.2.5 Modelo de gestão de memória

Define o particionamento, configuração e protecção da memória. Especifica ainda como é feita a tradução da memória, define instruções especiais de controlo de memória, e determina ainda outras características da gestão de memória.

3.2.2.6 Modelo de gestão temporal

Define recursos que permitem que a hora do dia seja determinada e os recursos e mecanismos necessários para suportar excepções relacionadas com o tempo.

3.2.3 Níveis da arquitectura *PowerPC*

Os aspectos referidos anteriormente da arquitectura *PowerPC* são definidos em 3 níveis. Esta divisão em níveis proporciona flexibilidade ao permitir diferentes graus de compatibilidade

de *software* numa ampla gama de implementações. Podemos considerar, por exemplo, uma implementação de um controlador embutido pode suportar o conjunto de instruções do utilizador, mas não a gestão de memória, excepções e modelos de cache.

Os três níveis são os seguintes:

3.2.3.1 **Arquitectura do conjunto de instruções do utilizador (UISA)**

- Define o nível da arquitectura a que o *software* ao nível do utilizador deve obedecer.
- Define o conjunto básico de instruções e os registos ao nível do utilizador, tipo dos dados, convenções de memória de vírgula-flutuante, modelo de excepção visto pelos aplicações do utilizador, modelo de memória e modelo de programação.

Todas as implementações *PowerPC* respeitam o UISA.

3.2.3.2 **Arquitectura de ambiente virtual (VEA)**

- Define funcionalidades adicionais ao nível do utilizador, que não se enquadram nos requisitos habituais do *software* ao nível do utilizador.
- Descreve o modelo de memória para um ambiente em que vários dispositivos tentam aceder à memória.
- Define aspectos do modelo de cache e instruções de controlo de cache.
- Define os recursos relacionados com a gestão temporal do ponto de vista do utilizador.

As implementações que respeitam o VEA respeitam igualmente o UISA

3.2.3.3 **Arquitectura de ambiente operativo (OEA)**

- Define recursos ao nível do supervisor, normalmente necessários por um sistema operativo.
- Define o modelo de gestão de memória, registos ao nível do supervisor, requisitos de sincronização e o modelo de excepção.
- Define os recursos relacionados com a gestão temporal do ponto de vista do supervisor.

As implementações que respeitam o OEA respeitam igualmente o UISA e o VEA.

Qualquer arquitectura *PowerPC* tem que respeitar o UISA, oferecendo total compatibilidade entre as aplicações para *PowerPC*, todavia podem existir diferentes versões do VEA e do OEA.

Apesar de a arquitectura definir alguns parâmetros para garantir compatibilidade, também permite uma ampla gama de opções para implementações personalizadas.

3.2.4 Arquitectura *PowerPC* para sistemas embutidos

Como o nome indica, esta arquitectura está optimizada para controladores embutidos, fornecendo algumas definições diferentes para certas características definidas pela VEA e OEA. Esta arquitectura é precursora da arquitectura *PowerPC Book-E*. Como esta arquitectura é de 32 bit não inclui as extensões para 64 bit presentes no UISA, em adição a estas características o suporte a vírgula-flutuante pode ser implementado em *hardware* ou *software*.

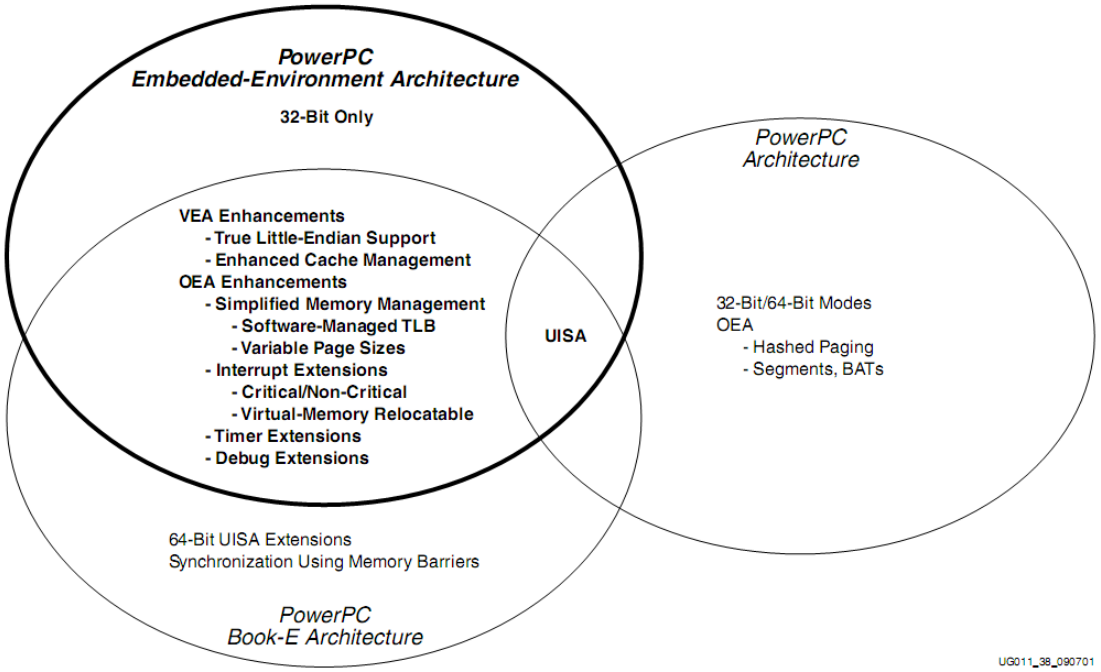

Figura 3.2 - Relação das arquitecturas *PowerPC* [5]

A arquitectura *PowerPC* para sistemas embutidos tem as seguintes características:

- Gestão de memória optimizada para sistemas embutidos.
- Instruções de gestão da cache para optimizar desempenho e controlo de memória em aplicações complexas que são graficamente e numericamente intensivas.
- Atributos de armazenamento para controlo do comportamento do sistema de memória.
- Registos especiais usados para controlar o uso dos recursos de depuramento*, timer*, interrupções e outros.
- Um espaço para endereços de registo de controlo de dispositivos para gerir periféricos *on-chip* como controladores de memória.
- Uma estrutura de interrupções de dois níveis e instruções para controlo de interrupções.
- Múltiplos recursos para timers.
- Recursos usados para depuramento que permitem funções de depuramento em *hardware* e *software* tais como *breakpoints* de instruções, *breakpoints* de dados e execução de programas passo-a-passo.

3.2.4.1 **Ambiente Virtual**

O ambiente virtual define características da arquitectura que permitem que aplicações possam criar ou modificar código, gerir coerência de armazenamento e optimizar o desempenho de acesso à memória. Define também os modelos de cache e memória, os recursos de gestão de tempo do ponto de vista do utilizador e recursos que estão acessíveis em modo de utilizador mas são principalmente usados em rotinas de bibliotecas do sistema.

3.2.4.2 **Ambiente operativo**

O ambiente operativo descreve características da arquitectura que permitem aos sistemas operativoss alocar e gerir o armazenamento, controlar os erros encontrados por aplicações, suportar dispositivos de I/O, e providenciar serviços de sistemas operacionais. Especifica os recursos e mecanismos que requerem acesso privilegiado, incluindo protecção de memória e mecanismos de tradução de memória, o modelo de tratamento de excepções e recursos privilegiados de *timer*.

3.3 PPC405

3.3.1 Características do PPC405

O processador PPC405 é uma implementação da arquitectura *PowerPC* para sistemas embutidos, fornecendo alto desempenho com baixo consumo energético sendo compatível com a arquitectura UISA. Algumas das principais características do PPC405 são:

- Uma unidade de execução de vírgula fixa totalmente compatível com a UISA,
  - Arquitectura de 32-bit, contendo trinta e dois 32-bit registos de uso geral (GPRs).
- Extensões da arquitectura *PowerPC* para sistemas embutidos fornecendo suporte adicional para aplicações em sistemas embutidos.
  - Funcionamento em verdadeiro *little-endian.*
  - Gestão flexível da memória.
  - Instruções de multiplicação com acumuladores para aplicações computacionalmente intensivas.
  - Capacidades de depuramento melhoradas.
  - Base temporal de 64-bit.
  - Três *timers*: *Timer* de intervalo programável (PIT), *timer* de intervalo fixo (FIT) e *watchdog timer*. (Todos síncronos com a base temporal).
- Características para melhoramento do desempenho, entre elas:
  - Previsão estática de *branch.*
  - *Pipeline* de 5 fases, sendo as instruções na sua grande maioria de um só ciclo,

incluindo *loads* e *stores*.

- Divisão e multiplicação em *hardware* para uma aritmética de inteiros mais rápida (4 ciclos para a multiplicação e 35 ciclos para a divisão).
- Tratamento de *strings* e várias palavras melhorado.
- Suporte a *loads* e *stores* desalinhados para *arrays* de cache, memória principal e memória *on-chip* (OCM)
- Interrupções com latência minimizada

- Cache de instruções integrado:
  - 16 KB, *2-way set associative.*
  - Oito palavras (32 bytes) por *cacheline.*
  - *Fetch line buffer.*
  - *Prefetch* programável da próxima linha sequencial para o *fetch line buffer.*
  - *Prefetch* programável de instruções *non-cacheable*: linha inteira (8 palavras) ou meia linha (4 palavras).
  - Não bloqueia durante o preenchimento de *fetch line.*
- Cache de dados integrado:
  - 16 KB, *2-way set associative.*
  - Oito palavras (32 bytes) por *cacheline.*
  - Leitura e escrita de *line buffers.*
  - *Load* e *store hits* são fornecidos de/para os *line buffers.*
  - Suporte a *write-back e write-through.*
  - *Load* e *store* programável.
  - Encaminhamento de operandos durante preenchimento da *cacheline.*
  - Não bloqueia durante o preenchimento de *fetch line.*
- Suporte para memória *on-chip* (OCM) com desempenho de acesso à memória idêntico a uma cache hit.
- Gestão de memória flexível:
  - Tradução de 4GB de espaço de endereços lógico em espaço de endereços físico.
  - Controlo independente sobre tradução e protecção de instruções e tradução e protecção de dados.
  - Controlo ao acesso ao nível de paginamento usando o mecanismo de tradução.
  - Controlo em *software* sobre a estratégia de substituição de páginas.
  - Controlo de atributos de armazenamento WIU0GE (*write-through, cacheability, user definined 0, guarded, endian*) para cada região de memória virtual.
  - Controlo de atributos de armazenamento WIU0GE para 32 regiões de 128 MB em modo real.
  - Protecção de controlo adicional usando zonas
- Suporte a depuramento melhorado com operadores lógicos
  - Quatro comparadores de endereços de instruções.
  - Dois comparadores de endereços de dados.
  - Dois comparadores de valores de dados.
  - JTAG para escrita na cache de instruções.

- *Tracing* de instruções para a frente e para trás.
- Gestão energética avançada.

3.3.1.1 **Privilégios**

O *software* que corre no PPC405 pode correr de dois modos diferentes, privilegiado e de utilizador. No modo privilegiado é permitido o acesso dos programas aos registos e a execução de todas as instruções suportadas pelo processador. Normalmente, o sistema operativo e os *device drivers* de baixo nível correm neste modo. No modo de utilizador, o acesso a alguns registos e instruções está restringido, por norma são os programas que correm neste modo.

3.3.1.2 **Modos de tradução de endereços**

O PPC405 suporta 2 modos de tradução de endereços: o real e o virtual. No modo real, os programas endereçam a memória física directamente. No modo virtual, os programas endereçam memória virtual e posteriormente estes endereços são traduzidos pelo processador em endereços de memória física. Isto permite aos programas aceder a um espaço para endereços maior do que o implementável pelo sistema.

3.3.1.3 **Modos de endereçamento**

Quer o PPC405 esteja a correr em modo real ou virtual, o endereçamento dos dados é suportado por instruções de *load* e *store* usando um dos seguintes modos de endereçamento:

- Registo indirecto com índice imediato – Um endereço de base é armazenado num registo, e o deslocamento desse endereço é especificado como um valor imediato na instrução.
- Registo indirecto com índice – Um endereço base é armazenado num registo, e o deslocamento desse endereço é armazenado num segundo registo.
- Registo indirecto – O endereço dos dados é armazenado num registo.

Instruções que usam os dois primeiros modos também permitem actualizações automáticas ao registo do endereço base. Nestes dois modos, o novo endereço dos dados é calculado, usado no acesso aos dados do tipo *load* ou *store* e armazenado no registo do endereço base.

Na execução sequencial de instruções, o endereço da instrução seguinte é calculado adicionando quatro bytes ao endereço da instrução actual. No caso de serem instruções *branch*, o endereço é calculado usando um dos seguintes 4 métodos.

- *Branch to relative* – O endereço da instrução seguinte está numa localização relativa ao endereço da instrução actual.
- *Branch to absolute* – O endereço da instrução seguinte está numa localização não relacionada com a instrução actual.
- *Branch to link register* – O endereço da instrução seguinte está armazenado no registo de ligação.
- *Branch to count register* – O endereço da instrução seguinte está armazenado no registo contador.

3.3.1.4 **Tipos de dados**

As instruções do PPC405 podem ter operandos do tipo byte, meia-palavra (16 bit) e palavra (32 bits). Os operandos com várias palavras são suportados pelas instruções múltiplas de *load*/*store* e as *strings* de bytes são suportados pelas instruções de *load*/*store* para *strings*. Os inteiros podem ter ou não sinal, e inteiros com sinal são representados usando o formato de complemento para dois.

O endereço de um operando com vários bytes é determinado usando o menor endereço de memória ocupado por esse operando.

3.3.1.5 **Registos**

Podemos observar com algum detalhe, na figura 3.4, os registos existentes nesta arquitectura

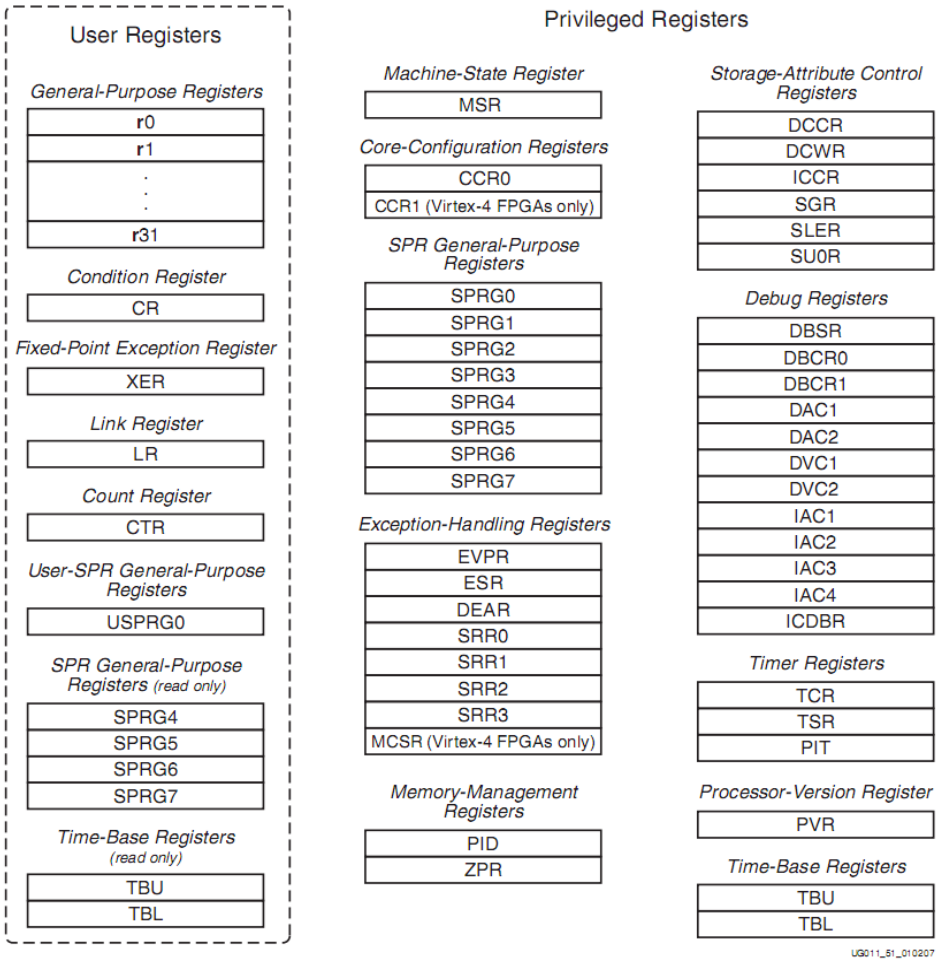

Figura 3.3 - Registos do *PowerPC*405 [5]

3.3.1.6 Organização do PPC405

O PPC405 contém seguintes elementos:

- *Pipeline* de 5 fases, com fases de fetch, *decode*, *execute*, *write-back* e *load write-back*.
- Gestão de memória virtual que suporta vários tamanhos de páginas e uma variedade de atributos de protecção de armazenamento e opções de acesso ao controlo.
- Unidades separadas de instruções e de cache.
- Suporte a debug, incluindo uma interface JTAG.
- 3 Timers programáveis.

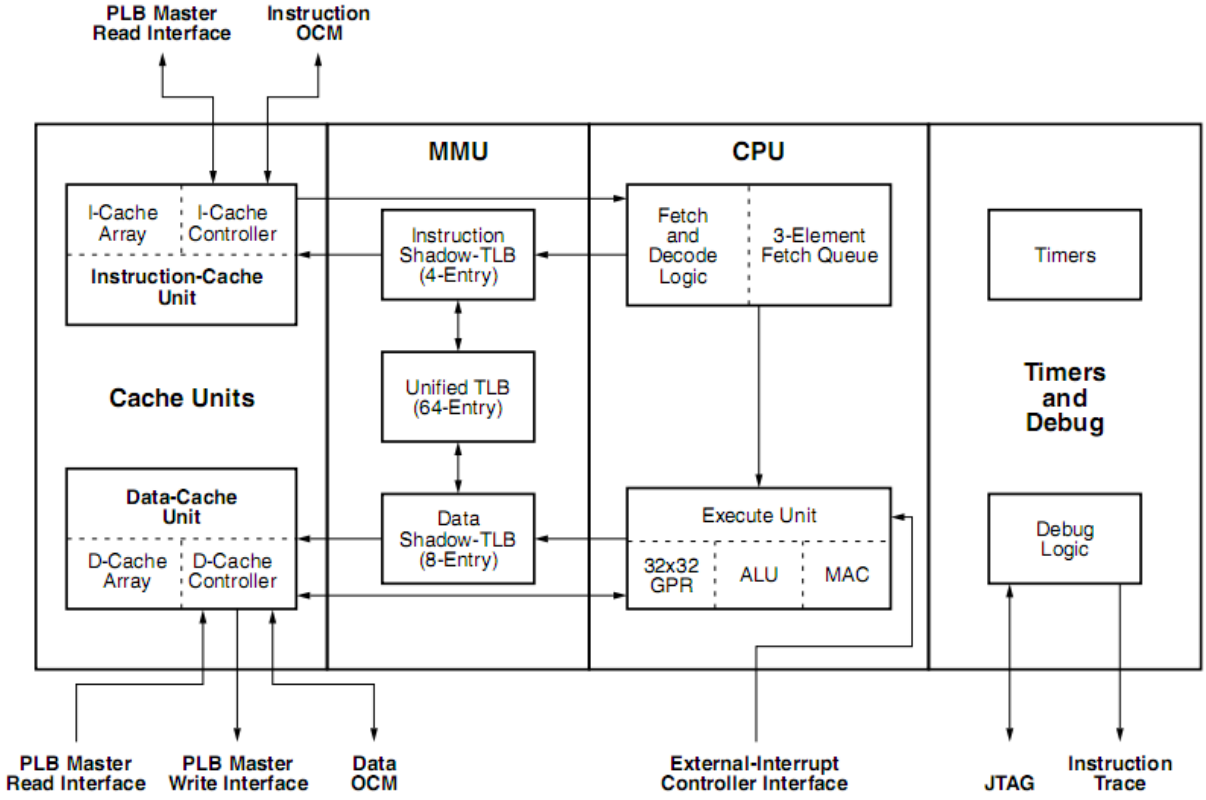

Figura 3.4 - Organização interna do PPC405 [5]

3.4 FX12

A placa *Nu Horizons* FX12 é uma plataforma de desenvolvimento de circuitos integrados para a Virtex-4 FX12 da *Xilinx*, sendo as suas principais características:

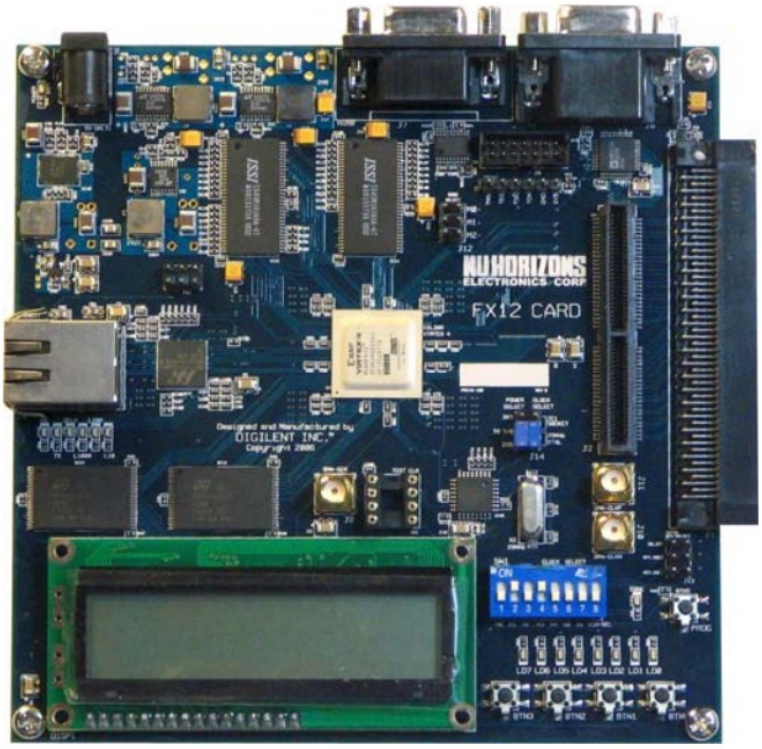

Figura 3.5 - FX12 Nu Horizons [3]

- FPGA Virtex-4 FX12
- Porta de programação JTAG
- Memória Flash para armazenamento de configurações da FPGA
- Porta Ethernet
- Saída para monitor VGA de 24 bit
- 64 MB de memória SDRAM
- 16 MB de memória Flash
- Sintetizador de frequência ajustável (até 350 MHz)
- Porta série RS-232
- Display LCD
- 4 Botões
- 8 Leds
- Conector de expansão de alta velocidade

A FX12 inclui um conjunto de características que trazem novas capacidades as plataformas embutidos, entre elas um PowerPC embutido, um MAC com hard-IP, *slices* XtremeDSP, multiplicadores em hardware rápidos, gestão avançada de relógio e Smart RAM.

A FX12 tem 32 *slices* XtremeDSP, 9 bancos de I/O, 648 Kb de Block RAM, 1368 CLBs e 4 DCMs [6]

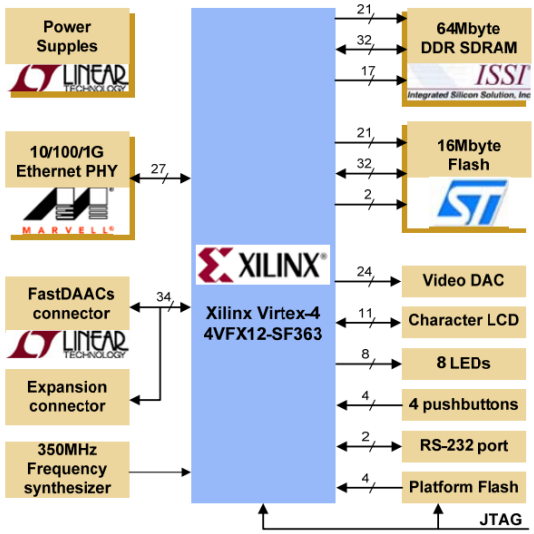

Figura 3.6 - Diagrama de blocos da FX12 [3]

3.5 Ferramentas de Apoio

3.5.1 *Xilinx Integrated Synthesis Environment* (ISE)

O *Xilinx* ISE consiste numa ferramenta que permite implementar, simular e programar qualquer FPGA ou CPLD da *Xilinx*, tudo isto a partir de um ambiente gráfico.

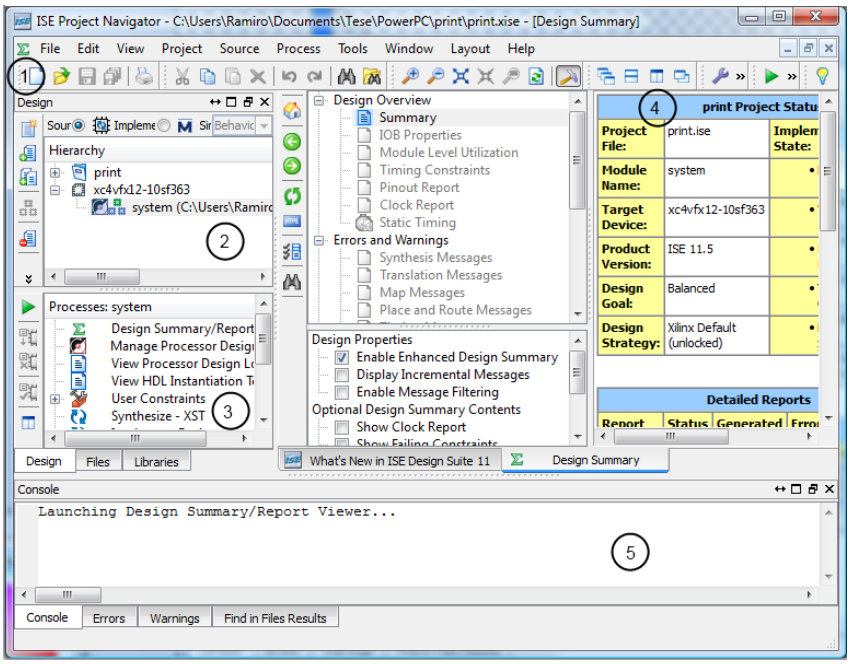

Figura 3.7 - Ambiente de trabalho do *Xilinx* ISE

Observando a figura 3.8 podemos dividir o ambiente de trabalho em cinco blocos.

O bloco 1, *Toolbar*, como em qualquer outro programa com interface gráfico, contém apenas alguns atalhos das funções mais usadas.

O bloco 2, *Sources Window*, mostra os ficheiros existentes no projecto e possibilita também a adição ou criação de novos ficheiros. Podemos também observar a hierarquia de um projecto. Por fim podemos também alterar algumas definições do projecto tais como a família do dispositivo, a ferramenta de síntese, a linguagem, etc.

O bloco 3, *Process Window*, permite correr "processos" no ficheiro seleccionado no bloco 2, estes mudam de acordo com o tipo de ficheiro seleccionado. Estes processos são mostrados de forma hierárquica, estando ordenados como um diagrama de fluxo de projecto normal, cada processo pode ter a frente uma variedade de sinais, que podem significar que o processo está a correr, que houve erros no processo, etc..

O bloco 4, *Workspace*, é onde aparecem os ficheiros VHDL, *Verilog*, etc., quando são abertos ou relatórios de implementação. Podem estar abertos diversas janelas mostrando diversos ficheiros, carregando no separador na parte de baixo traz o ficheiro pretendido para a frente.

O bloco 5, Transcript Window, mostra mensagens dos projectos que estão a correr, como erros de síntese ou avisos, quando aparece uma destas mensagens podemos ser direccionados para ela no caso de ser um erro de sintaxe ou aviso, ou podemos ser redireccionados para a internet de modo a obtermos soluções para o erro [10].

3.5.1.1 **Criação de um Projecto**

A criação de um projecto para uma FPGA ou um CPLD na *Xilinx* ISE segue vários passos sequenciais. Esses passos podem ser vistos na figura 3.9.

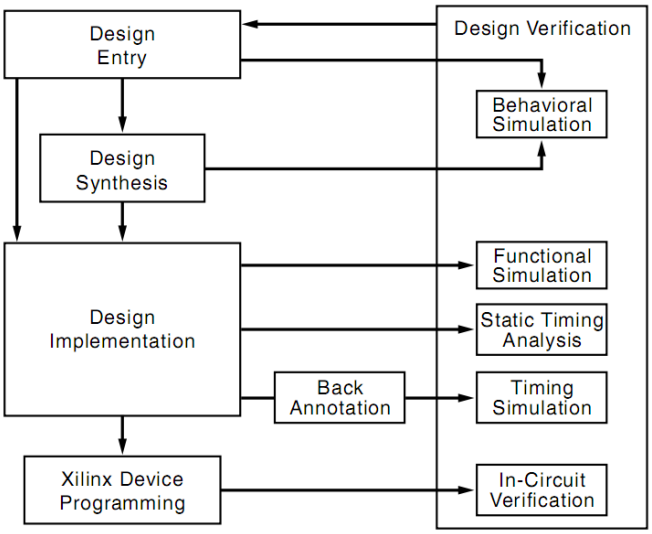

Figura 3.8 - Diagrama de fluxo do *Xilinx* ISE [4]

- *Design Entry* – A criação de um projecto passa pela inclusão no mesmo de ficheiros VHDL, esquemáticos, blocos das bibliotecas IP, e ficheiros UCF. Os ficheiros UCF incluem restrições ao nível temporal, de área ou de ligação de pinos. Depois desta fase, segue-se uma fase de síntese, ou uma fase de simulação comportamental.
- *Design Synthesis* – Depois da fase inicial, é preciso sintetizar o projecto. A primeira fase consiste numa detecção de erros de sintaxe, em seguido detectam-se erros de implementação. Por fim depois de resolvidos todos os erros, é criado um ficheiro chamado *netlist*, que serve de base para a etapa seguinte.
- *Design Implementation* – Nesta etapa, o ficheiro criado anteriormente é convertido num projecto lógico, que irá criar um ficheiro físico, podendo ser enviado para uma FPGA ou um CPLD. Esta fase pode ser dividida em 3 partes: *Translate, Map*, e *Place and Route*. Na primeira fase, *Translate*, as indicações presentes no UCF são traduzidas para um ficheiro lógico. Em seguida na fase *Map*, o circuito lógico criado é repartido de forma a ser enviado para a FPGA, sendo divido pelos vários blocos disponíveis, entre eles CLBs e IOBs. Na última fase, *Place and Route*, são criadas as ligações entre os blocos CLB ou IOB.
- *Xilinx Device Programming* – Nesta fase, como o nome indica o projecto é carregado para a FPGA. Antes do envio, o ficheiro é convertido num *bistream*, ficheiro .bit, para que a FPGA ou o CPLD o aceitem. Depois de criado, torna-se necessário programar a FPGA, através de uma aplicação integrada no ISE, o *iMPACT*, e um cabo de programação.
- *Design Verification* – O teste de um projecto ocorre em diferentes fases ao longo da criação de um projecto. Podemos identificar os seguintes momentos de verificação de um projecto:

1. *Behavioral Simulation* – É a primeira simulação do sistema criado e verifica se a implementação efectuada corresponde ao pretendido. O código é simulado através de simuladores que conseguem exibir o valor de todos os sinais num determinado instante de tempo. Dentro do ISE já existe um simulador: ISE *Simulator*.
2. *Functional Simulation* – Acontece após a fase *Translate*, usando a informação dada por esta acerca da operação lógica do circuito implementado. Nesta altura, o projectista verifica a funcionalidade do seu projecto, e se não estiver de acordo com o

esperado é necessário alterar o código

3. *Static Time Analysis* – Ocorre após as fases *Map* e *Place and Route* e serve para verificar se existem atrasos temporais no projecto, indica também a frequência máxima de operação do circuito implementado.
4. *Timing Simulation* – Permite analisar o sistema tendo em conta os atrasos existentes nos sinais. Pode ser realizada usando os mesmos simuladores da *Behavioral Simulation*, sendo mais fiável que a mesma.
5. *In-Circuit Verification* – É o teste final, serve para verificar como o projecto se comporta no dispositivo pretendido. Testa o circuito sobre condições típicas de funcionamento.

3.5.2 *XILINX Embedded Development Kit* (EDK)

O *Xilinx* EDK consiste num conjunto de ferramentas e Propriedade Intelectual (IP) que permite desenhar sistema completo com um processador embutido para implementação numa FPGA da *Xilinx*.

O EDK tem dois componentes principais:

- *Xilinx Platform Studio* (XPS) – É o ambiente de desenvolvimento usado para desenhar a parte do *hardware* do processador embutido.
- *Software Development Kit* (SDK) – É complementar ao XPS, é usado para criação e verificação de aplicações em C/C++. Tem como base a ferramenta de desenvolvimento open-source *Eclipse*.

Existem também outros componentes como:

- IP em *hardware* para processadores embutidos da *Xilinx.*
- *Drivers* e bibliotecas para o desenvolvimento de *software* para sistemas embutidos.
- GNU compilador e depurador para *software* em C/C++ com destino a processadores *MicroBlaze* e *PowerPC*.
- Documentação e projectos-amostra.

3.5.2.1 **Criação de um projecto**

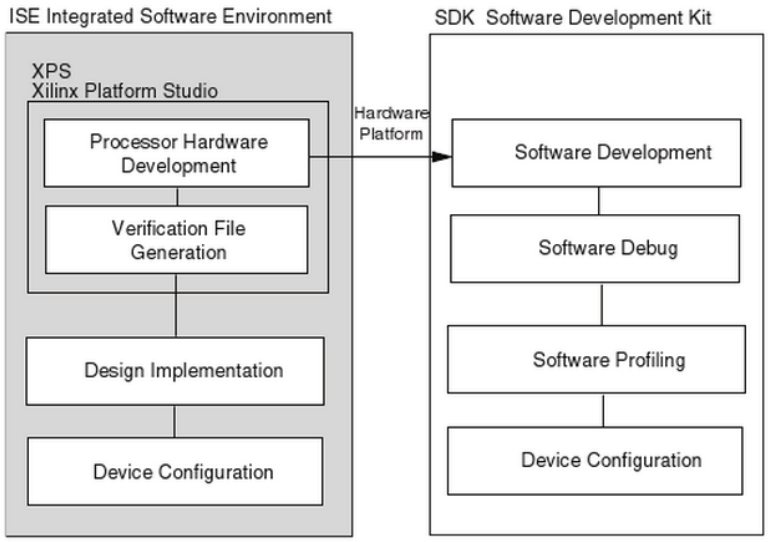

Figura 3.9 - Diagrama de fluxo do EDK[1]

Por norma, o ISE é usado para criar um processador embutido, que em seguida é adicionado ao projecto. Em seguida o projecto passa para o XPS onde é feita a especificação do microprocessador, dos periféricos assim como das ligações entre estes. O XPS permite verificar o correcto funcionamento da plataforma correndo o projecto num simulador. Esta simulação pode ser de 3 tipos:

- Behavioral
- Structural
- Timing-acurate

O próprio XPS facilita esta simulação fornecendo os ficheiros necessários à simulação automaticamente.

Após finalizar o projecto no XPS, é necessário voltar ao ISE para gerar os ficheiros de configuração da FPGA necessários para programar a mesma. O SDK é usado para desenvolvimento de *software*, após o depuramento deste *software* é criado um ficheiro *Executable and Linkable File* (ELF). Após a FPGA estar configurado com o ficheiro gerado no ISE podemos carregar o ficheiro ELF para a FPGA e acabar de depurar o mesmo.

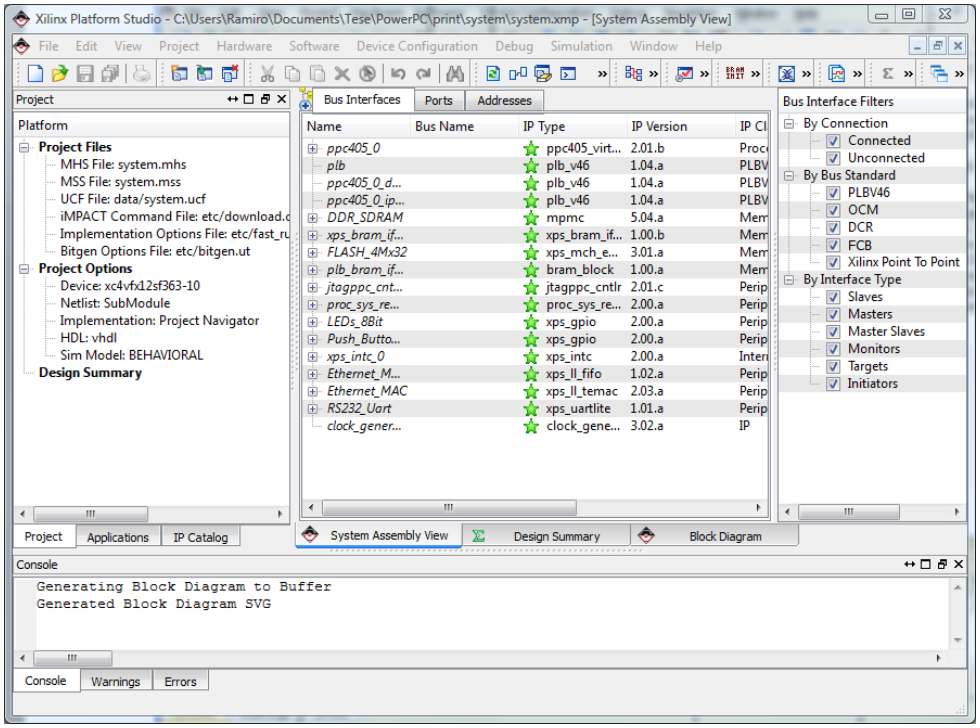

Figura 3.10 - Ambiente de Trabalho do *Xilinx* XPS

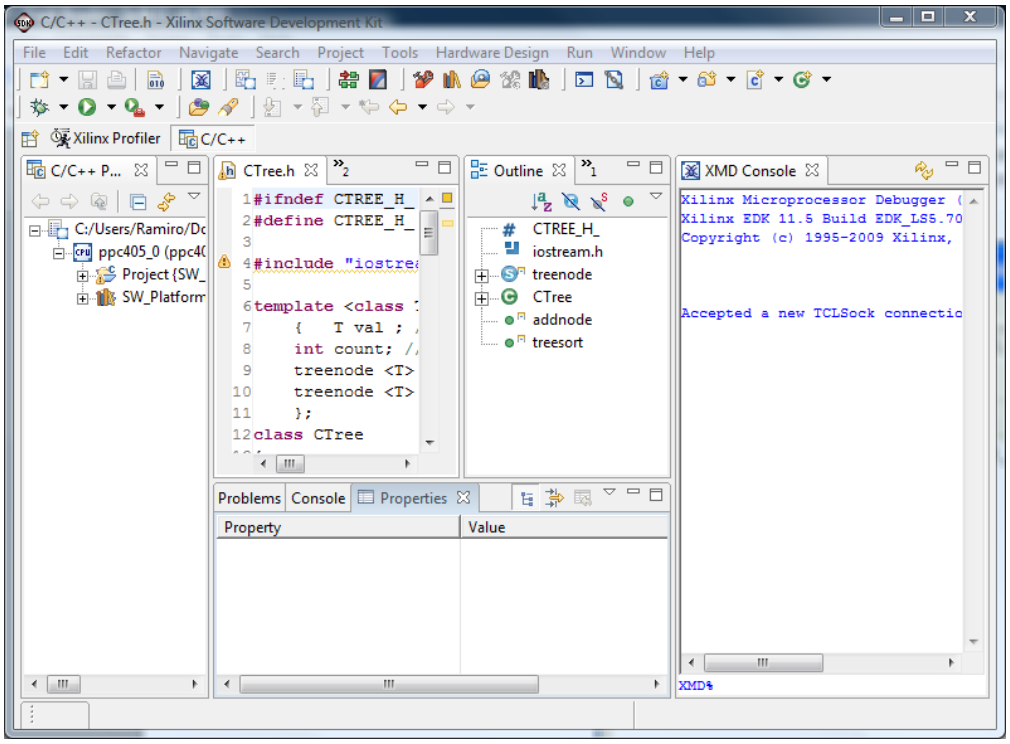

Figura 3.11 - Ambiente de Trabalho do *Xilinx* SDK

3.5.3 Microsoft Visual Studio

O Microsoft Visual Studio é um ambiente de desenvolvimento integrado (IDE) da Microsoft. Pode ser usado para desenvolver aplicações de consola e de interface gráfica, bem

como sites, aplicações Web, etc. Tem suporte a variadas linguagens de programação desde C/C++, C#, Python, etc.

Foi usado como método de comparação entre correr o código num sistema embutido e num PC.

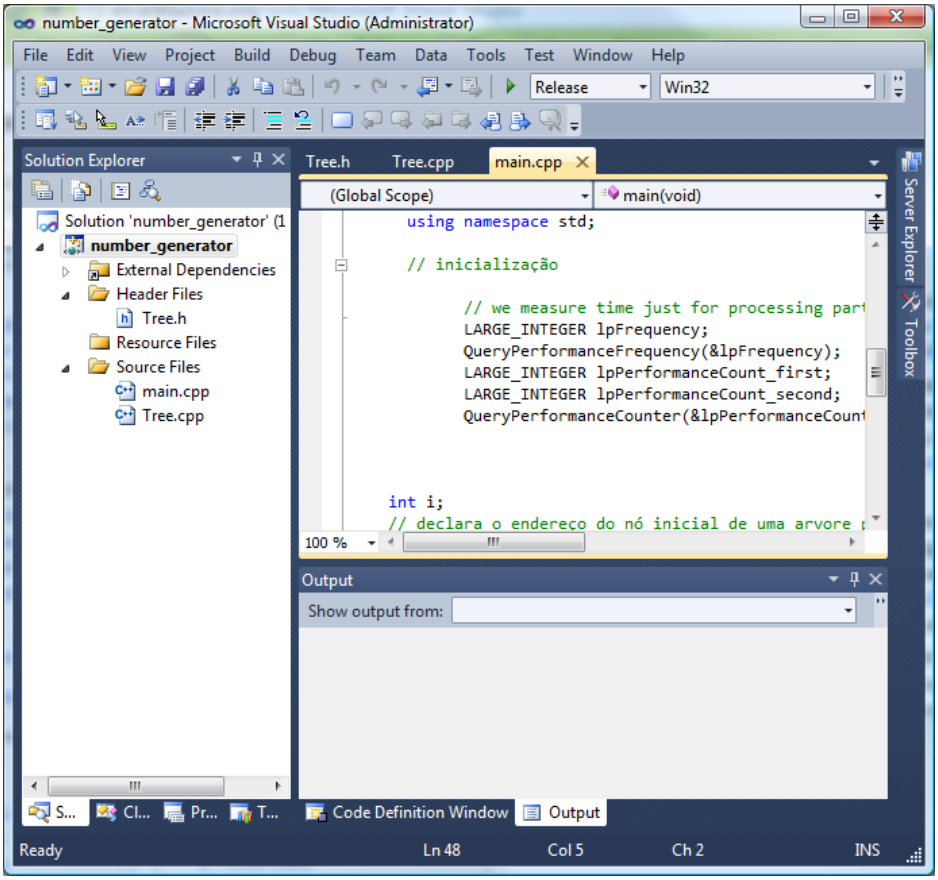

Figura 3.12 - Ambiente de trabalho do Visual Studio

3.5.4 PuTTY

O *PuTTY* é uma aplicação que serve como cliente para vários protocolos, entre eles o RS-232. Visto o Windows Vista não possuir nenhum cliente de origem para lidar com este protocolo, foi necessário recorrer ao mesmo. O protocolo RS-232 foi usado para comunicar com o PowerPC.

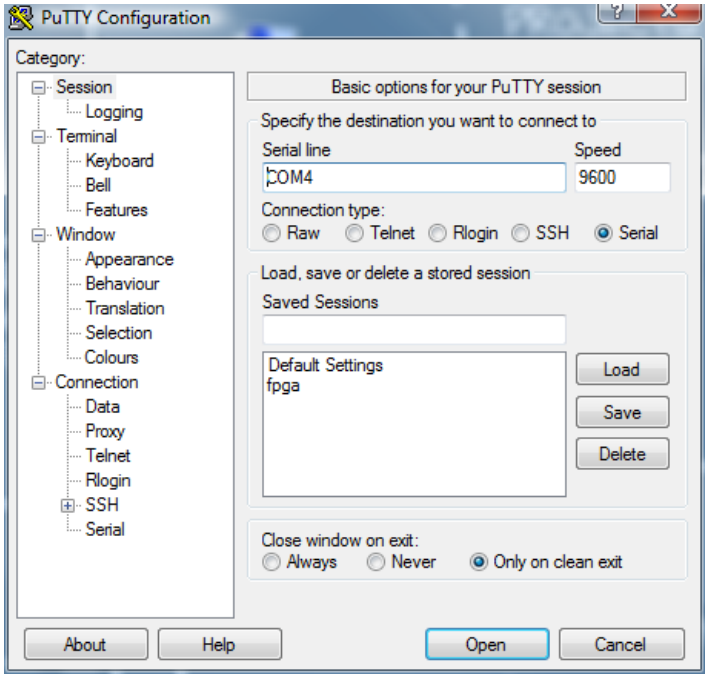

Figura 3.13 - Ambiente de trabalho do PuTTY

3.6 Conclusões

Neste capítulo foi feita uma introdução às ferramentas usadas neste trabalho, à arquitectura PowerPC e em particular ao PowerPC405, e as ferramentas de software usadas, Xilinx ISE, Xilinx EDK, Microsoft Visual Studio e PuTTY.

Capítulo 4 - *Implementação do algoritmo de ordenação de dados*

4.1 Introdução

A base deste trabalho era obter uma comparação entre a eficácia de diferentes métodos *Soft* e *Hard* quando aplicados em determinados algoritmos. Para tal foi escolhida a FPGA FX12 da *Nu Horizons*, que, como já foi dito, possui o chip Virtex4 da *Xilinx*, que por sua vez integra o processador *PowerPC* 405 da IBM.

4.2 Implementação em *PowerPC*

4.2.1 Especificação do *Hardware*

Para poder utilizar o processador embutido numa FPGA, há que primeiro fazer uma especificação do *hardware* que se quer utilizar. Para fazer esta especificação é usada uma ferramenta incluída no *Platform Studio*, o *Base System Builder*. No entanto antes de se poder usar esta ferramenta, é necessário criar um ficheiro do tipo *Xilinx Board Description* (XBD), visto que esta ferramenta não possui este ficheiro para a placa FX12, apenas para placas fabricadas pela *Xilinx*.

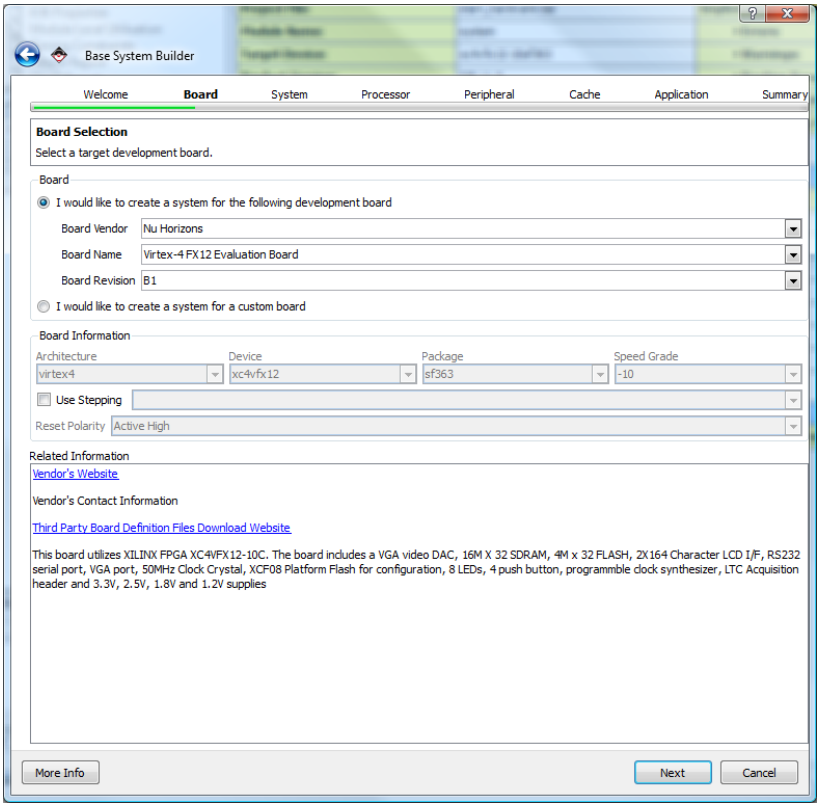

Figura 4.1 - Base System Builder

Durante este processo, é necessário escolher entre variadas opções, de modo a criar um projecto que possa ser usado na placa pretendida. Essas opções incluem:

- A placa a utilizar.
- O número de processadores a usar.
- O tipo de processador a usar e a sua frequência de relógio, entre outras.
- Os periféricos.
- A cache do processador.
- Opções para testes.

Neste caso as opções a serem seleccionadas são:

- *Nu Horizons* Virtex4 FX12 B1.
- Um único processador.
- *PowerPC*, a frequência de relógio assim como as outras opções não são relevantes agora, visto que irão ser alterados ao longo do trabalho.
- Dos periféricos disponíveis são necessários apenas: Memória SDRAM e Porta série RS-232.
- Na cache não é necessário alterar nada nas configurações da cache, nem nas opções para testes.

4.2.2 *Xilinx* Board Description - XBD

O ficheiro *Xilinx Board Description* (XBD) define o conteúdo de uma dada placa, e a sua interacção com a FPGA em si, este tipo de ficheiro tem as seguintes características:

- Dividido em blocos que definem os periféricos suportados pela placa.
- Cada bloco tem uma lista de atributos, parâmetros e portas.
- Informação da conectividade entre as diferentes portas e módulos.
- Informação sobre as restrições de cada pino da FPGA.

Neste projecto este tipo de ficheiro foi muito importante, pois sem ele seria impossível trabalhar com o processador embutido. No entanto devido ao facto de o fabricante já não dar suporte à placa, foi necessário pesquisar e adaptar os ficheiros existentes de placas semelhantes à usada neste projecto, sendo um processo bastante demorado, pois qualquer pequeno erro impossibilita que o projecto sequer funcione. Este ficheiro encontra-se no anexo A, visto não ser relevante a sua análise para compreensão do trabalho.

4.2.3 Desenvolvimento do *software*

Depois de criadas as especificações de *hardware*, avançou-se para o desenvolvimento do *software*. Nesta fase foi escolhido um algoritmo recursivo de ordenação baseado em máquinas de estado cujo diagrama de fluxo pode ser observado na figura 4.2.

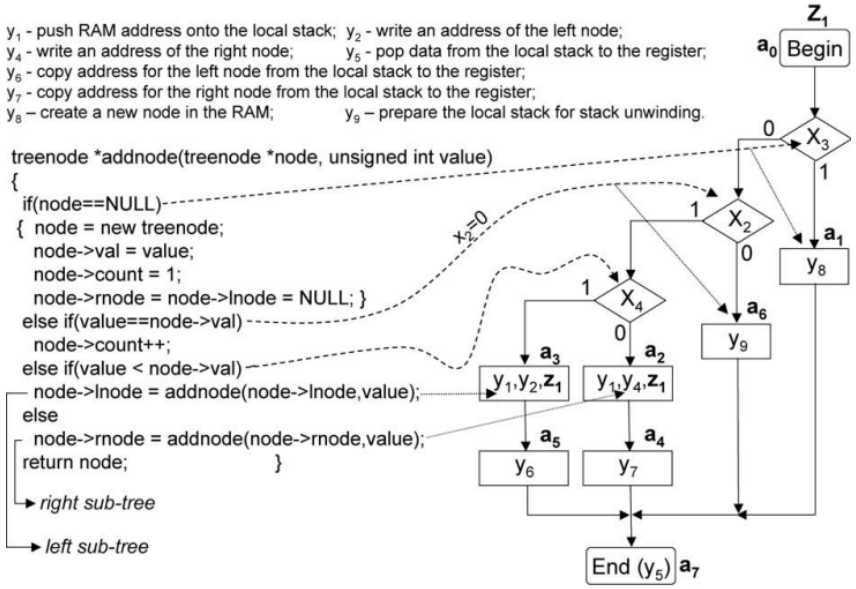

Figura 4.2 - Diagrama de fluxo da RHFSM [16]

Uma explicação muito resumida deste algoritmo é, quando o algoritmo recebe um valor vai começar por verificar se existe algum nó (nó esse que tem quatro campos, o valor, uma

variável contadora, e dois ponteiros para dois nós, um para um vector com um valor menor e outro com um valor maior). Se não existir vai criar um, atribuindo o valor recebido a esse nó, se já existir um nó vai verificar se o valor presente no nó é maior ou menor do que o valor recebido, colocando o valor para o nó presente no ponteiro conforme o valor.

4.2.4 Programação da placa

Após ser feito o depuramento com sucesso do *software*, é necessário enviá-lo para a placa, para tal é preciso seguir alguns passos:

- Programar a placa com um ficheiro do tipo *bitstream* (BIT) que contém a configuração do *hardware* da placa e com um ficheiro do tipo *Block Memory Map* (BMM) que é usado para carregar a memória da placa.
- Criar um ficheiro do tipo *Executable and Linkable File* (ELF). Para tal é necessário definir o tamanho, o tipo de memória e posição onde vão ser colocados alguns ficheiros.

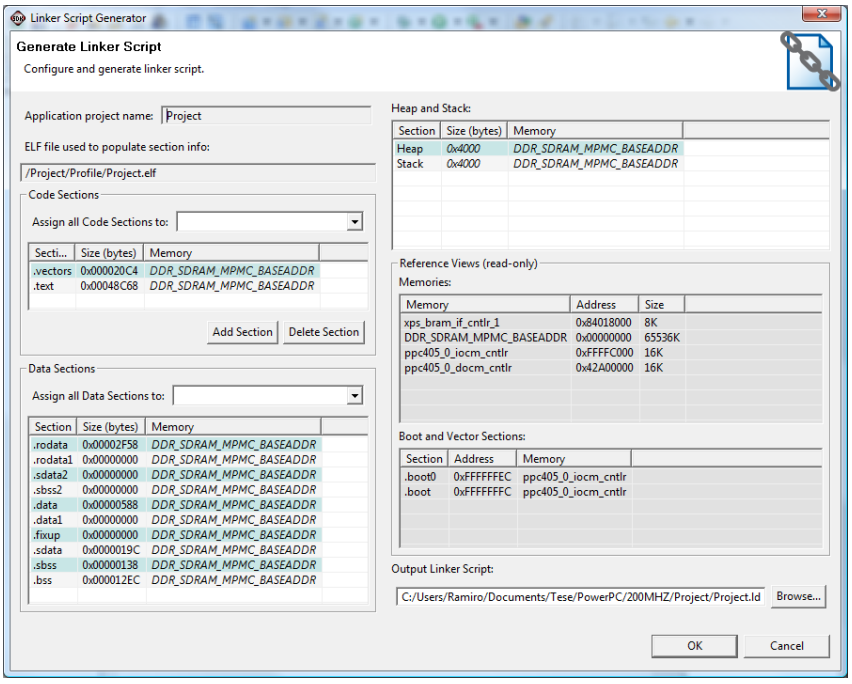

Figura 4.3 - Gerador do Executable and Linkable File

Após estes passos é possível programar a placa com o ficheiro ELF. Existem duas opções, é possível correr o programa sem interrupções, ou é possível correr o programa em modo depuramento, onde, naturalmente, é possível verificar se o programa está a correr sem erros e identificar esses erros.

4.2.5 *Profiling*

O método usado para obter informações acerca do tempo de execução dos programas no processador embutido é conhecido por *Profiling*. Este método consiste em alterar o modo como o programa corre para obter dados acerca dele, como tal este método é considerado como *software intrusive*. A normal execução do programa é alterada de duas maneiras:

- Para obter dados para um histograma, o programa é interrompido periodicamente para identificar que função está a correr. Essa periodicidade é definida pelo utilizador.
- Para saber quais as funções que foram chamadas e quantas vezes, é anotada também cada vez que uma função é chamada [2] [8].

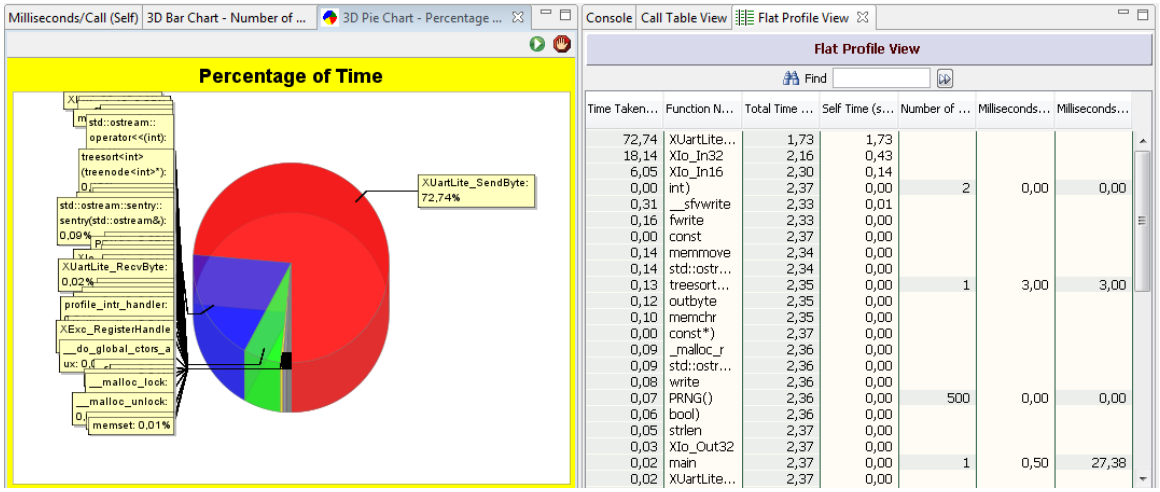

Figura 4.4 - *Profiling* no SDK

4.2.6 Visualização dos Resultados

Apesar de não ser relevante para o projecto, para confirmar que o programa está a trabalhar correctamente, foi usado, como já referido, o *PuTTY*. É necessário introduzir algumas configurações no programa, como:

- A porta onde vão ser lidos os resultados
- A *baud rate*
- O número bits de dados
- O número de *stop bits*
- Neste caso, a porta é a COM8, a *baud rate* é 9600, o número de bits de dados é oito, e há um stop bit.

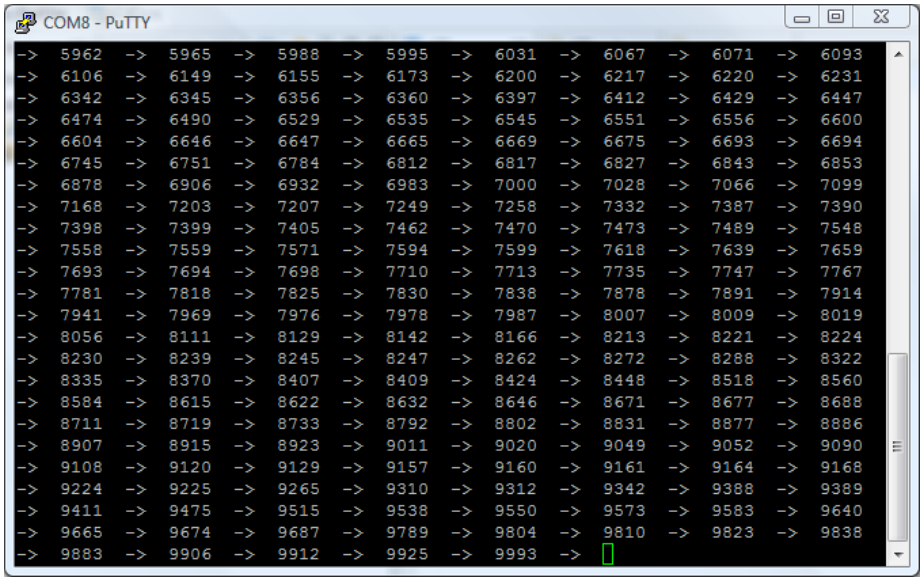

Figura 4.5 - Resultados vistos no PuTTY

4.3 Implementação na FPGA

As especificações que foram usados na FPGA, já existiam, foram só alterados de modo a correr na placa FX12, visto terem sido criados para correr na Nexys-2.

4.3.1 Máquinas de Estados Finitos

Todos os algoritmos utilizados neste trabalho tem como base uma máquina de estados finitos, como tal torna-se necessário compreender como funcionam e em que consistem.

4.3.2 Máquina de estados finitos simples (FSM)

Uma máquina de estados finitos (FSM) é definida, em termos gerais, por um conjunto de entradas, um conjunto de saídas, um conjunto de estados, uma função de transição que regula transições entre estados e uma função de saída, sendo os estados apenas valores abstractos que impõem passos numa sequência de operações. Estas máquinas receberam o seu nome devido ao facto de o conjunto de estados ser finito em tamanho. Por cada ciclo do relógio a máquina está apenas em um estado. A função de transição determina o próximo estado a ser implementado no ciclo de relógio seguinte, com base no estado actual, e possivelmente, as entradas no dado ciclo de relógio. A função de saída determina os valores das saídas num dado ciclo de relógio com base no estado actual e, por norma, também baseado nas entradas num dado ciclo de relógio.

Por norma as máquinas de estado dividem-se em dois tipos:

- Máquinas de Mealy
- Máquinas de Moore

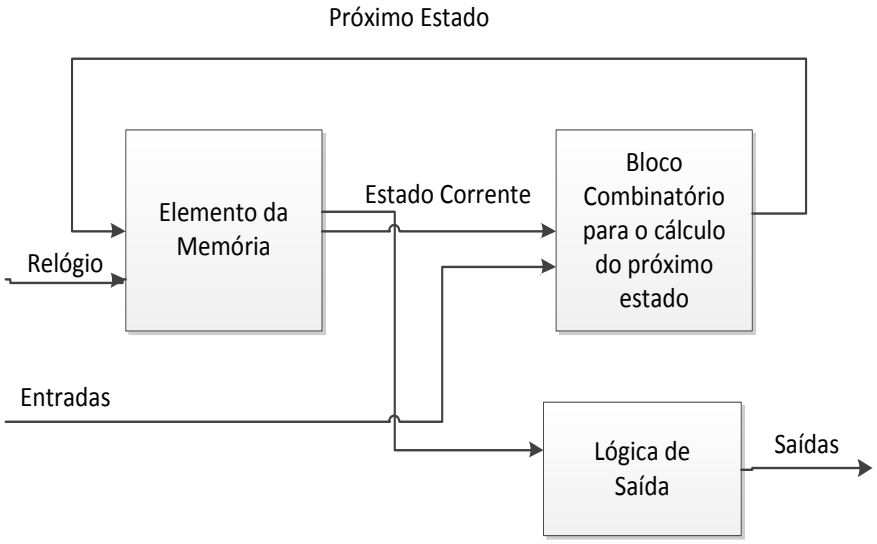

Figura 4.6 - Máquina de Moore

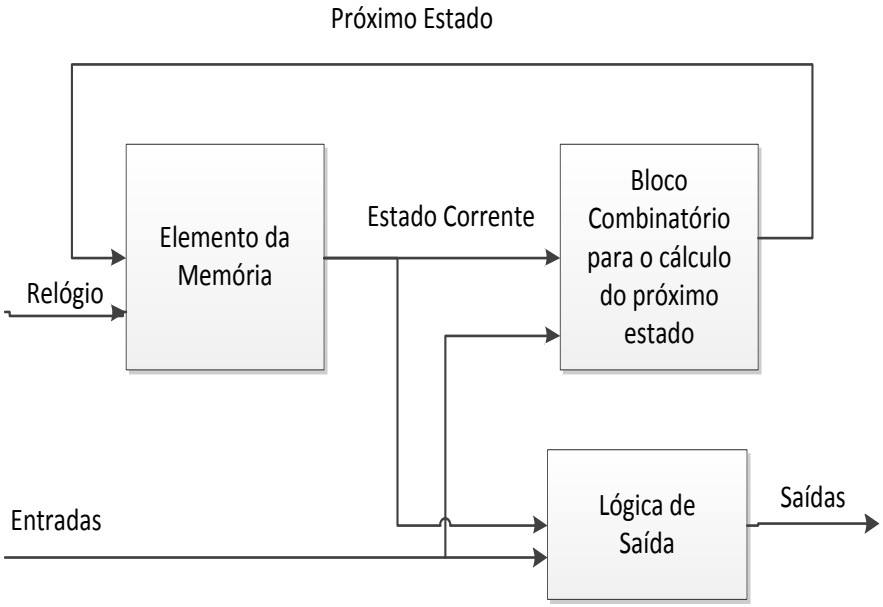

Figura 4.7 - Máquina de Mealy

Numa máquina de Mealy, a função de saída depende tanto do estado actual, como dos valores das entradas. Já na máquina de Moore a função de saída depende apenas do estado actual, e não dos valores de entrada. Na teoria podemos dizer que para cada máquina de Moore, existe uma máquina de Mealy, e o contrário também se aplica, no entanto, na prática apenas um ou o outro tipo é apropriado para o caso. Uma máquina de Mealy pode ser capaz de

implementada com menos estados, no entanto, pode ser incapaz de cumprir as restrições temporais, devido a atrasos na chegada das entradas, usadas para calcular as saídas [15].

4.3.3 Alterações feitas

A primeira alteração a ser feita, foi nas propriedades do projecto.

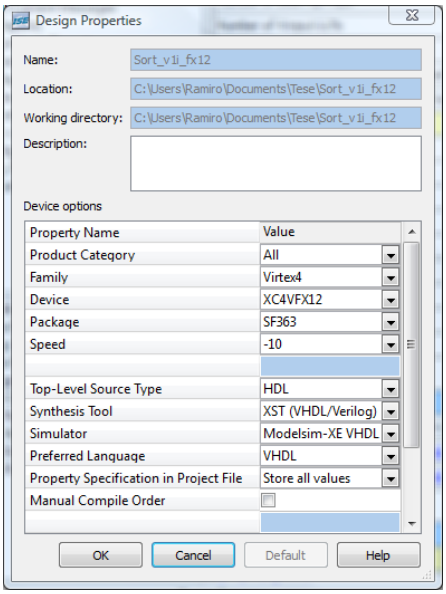

Figura 4.8 - Propriedades do Projecto

Em seguida, é necessário mudar o ficheiro com as restrições da placa, UCF.

As últimas alterações são relacionadas com o sinal VGA das duas placas, enquanto por exemplo a Nexys2, usa um sinal RGB com 8 bit (2 para o azul, 3 para vermelho e 3 para verde), o da FX12 é de 24bit (8 para azul, 8 para vermelho e 8 para verde). As outras diferenças podem ser vistas nas figuras 4.7 e 4.8, que representam os conversores de imagem de digital para analógico de cada placa.

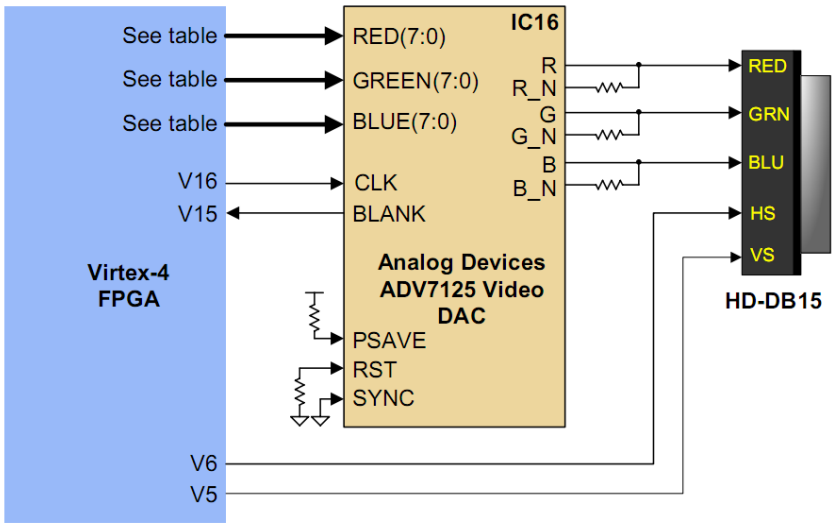

Figura 4.9 -FX12 DAC [3]

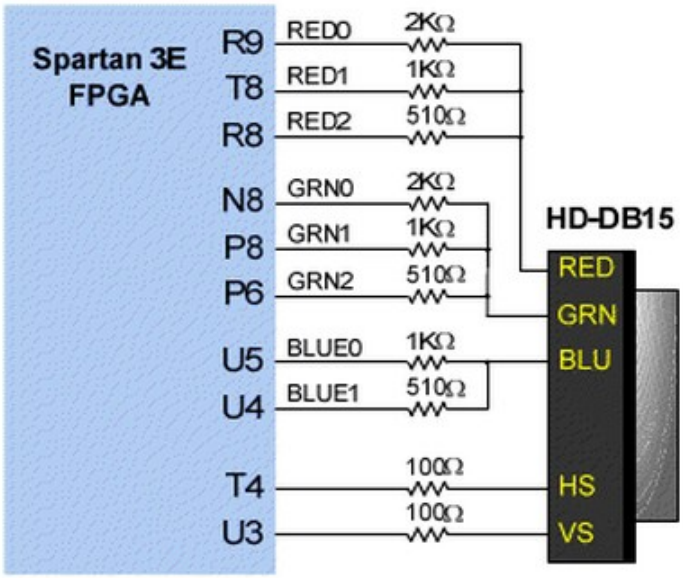

Figura 4.10 - Nexys2 DAC [22]

4.4 Implementação em Microsoft Visual Studio

A implementação no *Microsoft Visual Studio* serviu inicialmente para verificar o código usado no PPC405, visto o *Visual Studio* possuir um ambiente gráfico mais fácil e intuitivo de usar do que o *Xilinx* SDK. Numa fase posterior foi usado para comparação de resultados. O código utilizado está no Anexo B.

4.5 Recursividade

Como o algoritmo usado neste projecto tem como base a recursividade, será em seguida feita uma pequena introdução à mesma.

É sabido que a recursão é uma técnica de resolução de problemas extremamente poderosa, que permite que um problema seja decomposto em subproblemas mais pequenos, que são semelhantes ao problema original. No entanto esta técnica nem sempre é apropriada, sobretudo se existir uma solução iterativa mais eficiente. Isto deve-se ao elevado número de estados que são acumulados durante as chamadas recursivas. Além disso na maior parte das linguagens de alto nível, a chamada de uma função incorre num *overhead* dos registos. As funções recursivas aumentam esta sobrecarga, porque uma chamada inicial da função pode gerar um elevado número de invocações recursivas dessa função. No entanto a recursividade pode ser implementada mais eficientemente em *hardware*. Isto deve-se ao facto de que qualquer activação de uma subsequência recursiva de operações ter sido combinada com a execução de operações que são requeridas pelo respectivo algoritmo. O mesmo evento toma lugar quando qualquer subsequência recursiva está a ser terminada, isto é, quando o controlo foi devolvido ao ponto da última chamada recursiva e a operação do algoritmo a ser executado que segue a última chamada recursiva tem que ser activada. O número de estados requeridos para execução de recursão em *hardware* é reduzido quando comparado com *software*. Os estados são acumulados em pilhas normalmente implementadas em blocos de memória embutidas, que são relativamente baratos.

É sabido que um algoritmo recursivo pode ser implementado numa máquina de estados finita hierárquica (HFSM). No entanto, o modelo base de uma HFSM tem um número de desvantagens, incluindo um *overhead* do registo [16].

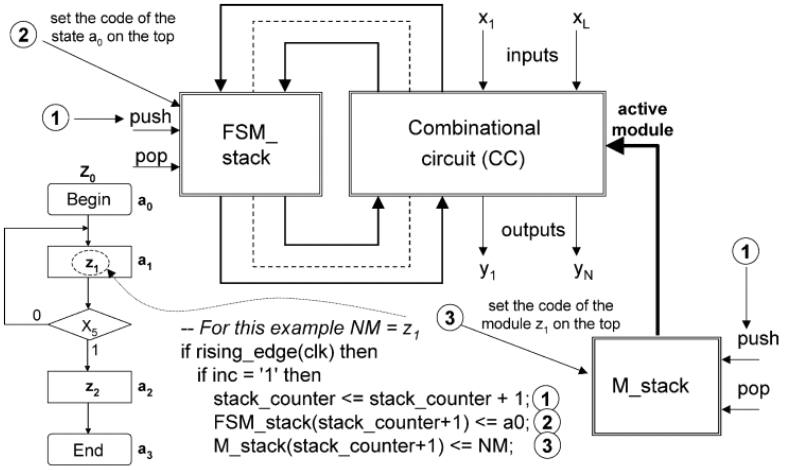

Figura 4.11 - Diagrama da RHFSM usada [16]

É de notar que apesar de os algoritmos recursivos terem imensas aplicações práticas, são usados mais frequentemente em vários tipos de pesquisa binária, sendo bastante eficientes nesta área, mesmo as soluções em *software.*

4.6 Conclusões

Neste capítulo foi referido o necessário para implementar o algoritmo nas várias plataformas, além de umas introduções a conceitos como máquinas de estados finitos e recursividade.

Capítulo 5 - *Resultados e Conclusões*

5.1 Introdução

Como já foi dito em capítulos anteriores, a área dos sistemas embutidos está em grande evolução, até aqui foram referidos as suas vantagens em variados aspectos e situações. Agora vai ser feita uma comparação entre várias plataformas em situações semelhantes.

5.2 Tecnologia Utilizada

A base deste trabalho é uma FPGA Virtex4-FX12 da *Nu Horizons*, com um processador embutido *PowerPC*405. Para comparação usou-se também um PC HP *EliteBook* 2730p com um processador Intel Core 2 Duo CPU @ 1.87 GHz e 4Gb de memória RAM. Para comparar estes sistemas usaram-se algoritmos baseados em máquinas de estados finitos.

Para criar os algoritmos para cada plataforma foram usados o *Microsoft Visual Studio* para o PC, o *Xilinx Embedded Development Kit* para o *PowerPC*405 e o *Xilinx Integrated Synthesis Environment* para a FPGA.

5.3 Metodologia Utilizada

Em relação ao PC e ao *PowerPC* o algoritmo utilizado foi o mesmo visto ambas as plataformas suportarem C/C++, como tal foi usado um código em C++, para a FPGA foram usados códigos diferentes com base em VHDL.

O algoritmo em C++ consiste num gerador de números aleatórios entre 0 e 10000 e numa máquina de estados finitos hierárquicas que ordena esses números. Para obtenção do tempo de execução no *PowerPC* foi usado um método chamado *Profiling*. Para o mesmo efeito no PC foi usada uma função incluída no *software* que conta o tempo de execução. A quantidade de números gerados nesta parte varia entre 5000 e 50000 para efeitos de comparação

Quanto à FPGA, devido ao facto de ser uma plataforma que usa uma linguagem diferente, o algoritmo usado foi implementado de maneiras diferentes, para uma melhor comparação. Os números gerados são também entre 0 e 10000, sendo no entanto à volta de 1200. Para obtenção do tempo de execução são contabilizados os ciclos de relógio e tendo conhecimento da frequência de relógio e calculado o tempo desejado.

Quanto às implementações do algoritmo elas diferem umas das outras em pormenores, descritos de seguida

5.3.1 Implementação 1

Como podemos ver na figura 5.1, esta implementação consiste na implementação descrita anteriormente em 4.2.3, diferindo apenas pela ausência da variável contadora, pelo que não vai ser descrita novamente.

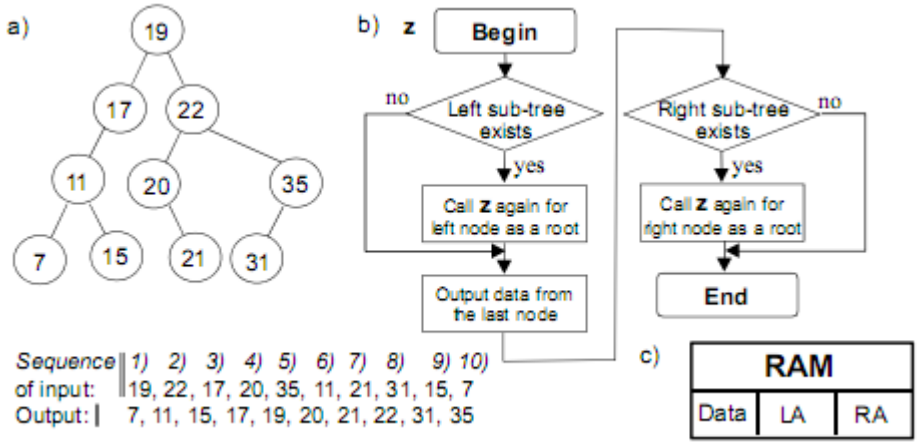

Figura 5.1- Implementação 1 a) Árvore Binária b) Algoritmo recursivo para ordenação c) Conteúdo da Memória [18]

5.3.2 Implementação 2

A implementação 2 difere da primeira implementação no uso de memórias *dual-port,* para armazenar palavras, tanto para a sub-árvore da esquerda e da direita dos nós e um *buffer* de registos para armazenar o nó seleccionado. Além disso usa um módulo, neste caso chamado z1, diferente do usado na implementação 1, este módulo pode ser visto na figura 5.2.

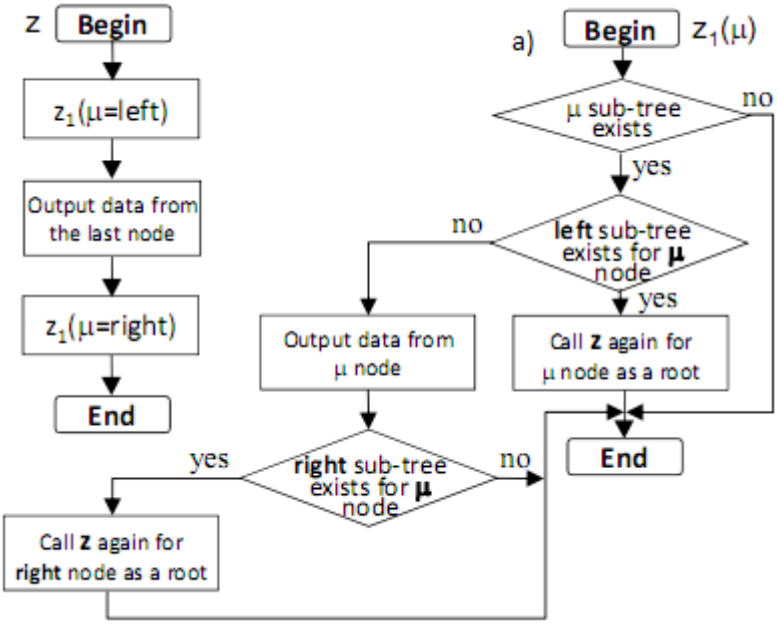

Figura 5.2 - Implementação 2 a) Módulo diferente [18]

5.3.3 Implementação 3

A implementação 3 é bastante semelhante a 2, a única diferença é que verifica LA e RA para cada palavra na memória *dual-port* independentemente.

5.3.4 Implementação 4

A implementação 4, consiste na implementação 3 mas usa o módulo antigo, e não o novo módulo usado na implementação 2 e 3.

5.4 Resultados da FPGA

	Frequência de Relógio (MHz)	Slices usadas	LUTs usadas	BRAMs usadas
Implementação 1	205,2	1915	3086	5
Implementação 2	213,4	1998	3176	6
Implementação 3	205,6	1987	3214	6
Implementação 4	207,5	1556	2686	5

Tabela 5.1 Comparação dos resultados dos algoritmos

Verificando estes dados podemos concluir que, o algoritmo 2 é o mais rápido em termos de frequência de relógio, todavia em termos de recursos utilizados (*Slices,* LUTs *e* BRAMs), o

algoritmo 4 é muito mais eficiente. Para uma melhor interpretação destes dados é necessário olhar agora para a tempo de execução de cada algoritmo em particular.

Ciclos de Relógio	Nº de dados	Tempo Total	Tempo por dado
4795	1199	23,4 µs	19,5 ns
4823	1206	23,5 µs	19,5 ns
5015	1254	24,4 µs	19,5 ns
5143	1286	25,1 µs	19,5 ns
5207	1302	25,4 µs	19,5 ns

Tabela 5.2 - Resultados da implementação 1

Ciclos de Relógio	Nº de dados	Tempo Total	Tempo por dado
7639	1216	35,8 µs	29,4 ns
8040	1245	37,7 µs	30,3 ns
8072	1265	37,8 µs	29,9 ns
8101	1286	38,0 µs	29,5 ns
8223	1304	38,5 µs	29,6 ns

Tabela 5.3 - Resultados da implementação 2

Ciclos de Relógio	Nº de dados	Tempo Total	Tempo por dado
6952	1193	33,8 µs	28,3 ns
7009	1226	34,1 µs	27,8 ns
7208	1262	35,1 µs	27,8 ns
7277	1267	35,4 µs	27,9 ns
7306	1273	35,5 µs	27,9 ns

Tabela 5.4 - Resultados da implementação 3

Ciclos de Relógio	Nº de dados	Tempo Total	Tempo por dado
3378	1210	16,3 µs	13,5 ns
3437	1236	16,6 µs	13,4 ns
3480	1249	16,8 µs	13,4 ns
3520	1265	17,0 µs	13,4 ns
3653	1320	17,6 µs	13,3 ns

Tabela 5.5 - Resultados da implementação 4

Olhando para estes dados podemos verificar que a implementação 4 é a mais eficiente, quer em termos de tempo total para cada conjunto de dados, quer em tempo por dado. Já a

implementação 2 apesar de conseguir ter a frequência de relógio mais elevado é também o que mais ciclos de relógio consome, e logo mais tempo demora. Como tal, vamos usar a implementação 4 para comparação com os resultados obtidos em *software* e em processador embutido.

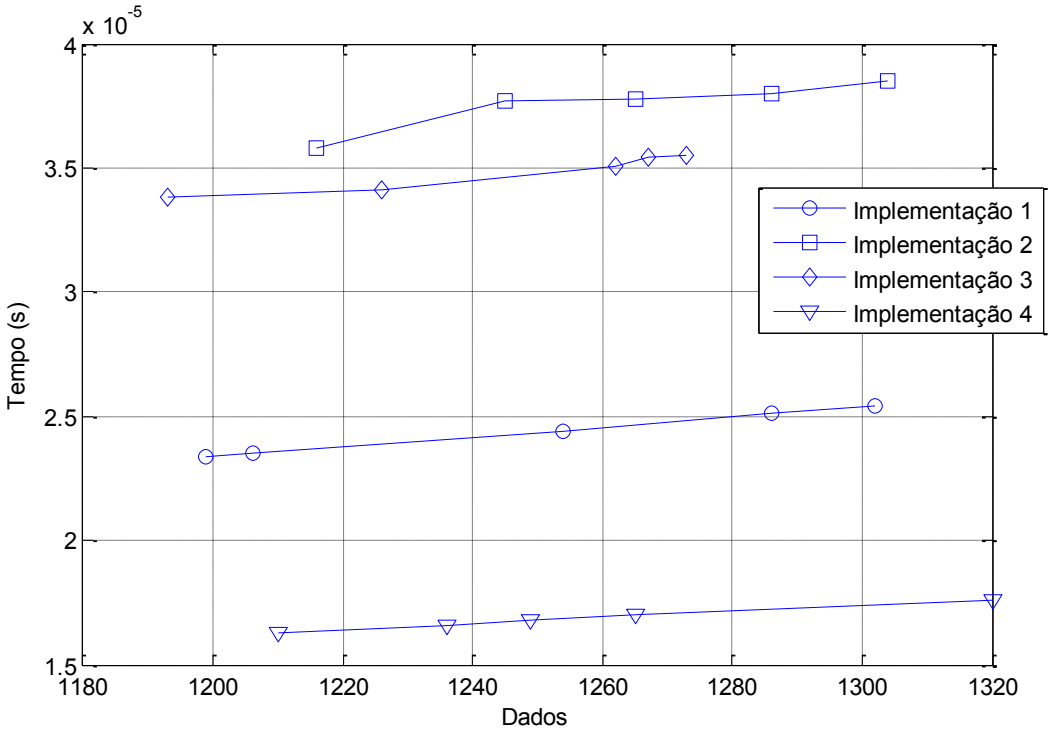

Figura 5.3 - Comparação temporal dos algoritmos usados

5.5 Resultados no PC

Nº de dados	5000	10000	20000	30000	40000	50000
Tempo	1,84 ms	2,90 ms	5,60 ms	7,90 ms	10,4 ms	12,0 ms
Tempo por dado	0,368 µs	0,29 µs	0,28 µs	0,263 µs	0,259 µs	0,24 µs

Tabela 5.6 - Resultados no PC

Como podemos verificar apesar de o PC ter uma frequência de relógio muito superior ao da FPGA obtemos resultados muito piores. Comparando podemos ver que os resultados obtidos na FPGA são melhores por uma ordem de grandeza.

5.6 Resultados no *PowerPC*

50 MHz

Nº de dados	5000	10000	20000	30000	40000	50000
Tempo	0,17 s	0,27 s	0,56 s	0,83 s	1,09 s	1,25 s
Tempo por dado	34 µs	27 µs	28 µs	27,7 µs	27,3 µs	25,0 µs

Tabela 5.7 - Resultados com processador a 50 MHz

100 MHz

Nº de dados	5000	10000	20000	30000	40000	50000
Tempo	0,2 s	0,39 s	0,78 s	1,18 s	1,56 s	2,01 s
Tempo por dado	40,0 µs	39,0 µs	39,0 µs	39,3 µs	39,0 µs	40,2 µs

Tabela 5.8 - Resultados com processador a 100 MHz

200 MHz

Nº de dados	5000	10000	20000	30000	40000	50000
Tempo	0,34 s	0,77 s	1,57 s	2,18 s	2,87 s	3,65 s
Tempo por dado	68,0 µs	77,0 µs	78,5 µs	72,7 µs	71,8 µs	73,0 µs

Tabela 5.9 - Resultados com processador a 200 MHz

300 MHz

Nº de dados	5000	10000	20000	30000	40000	50000
Tempo	0,51 s	1,09 s	2,3 s	3,44 s	4,08 s	5,22 s
Tempo por dado (s)	0,102 ms	0,109 ms	0,115 ms	0,115 ms	0,102 ms	0,104 ms

Tabela 5.10 - Resultados com processador a 300 MHz

Como podemos observar estes resultados não são satisfatórios, visto que com o aumento da frequência obtemos resultados piores que com frequências mais baixas. Uma possível causa para isto será o facto de o método usado para obter o tempo de execução ser um método *software intrusive*, isto é, intervém no normal funcionamento do *software*, parando-o para conseguir determinar qual a função que está a correr no momento. Além disto estes tempos são também piores que os tempos do algoritmo em *software* e que os dos algoritmos a correr na FPGA.

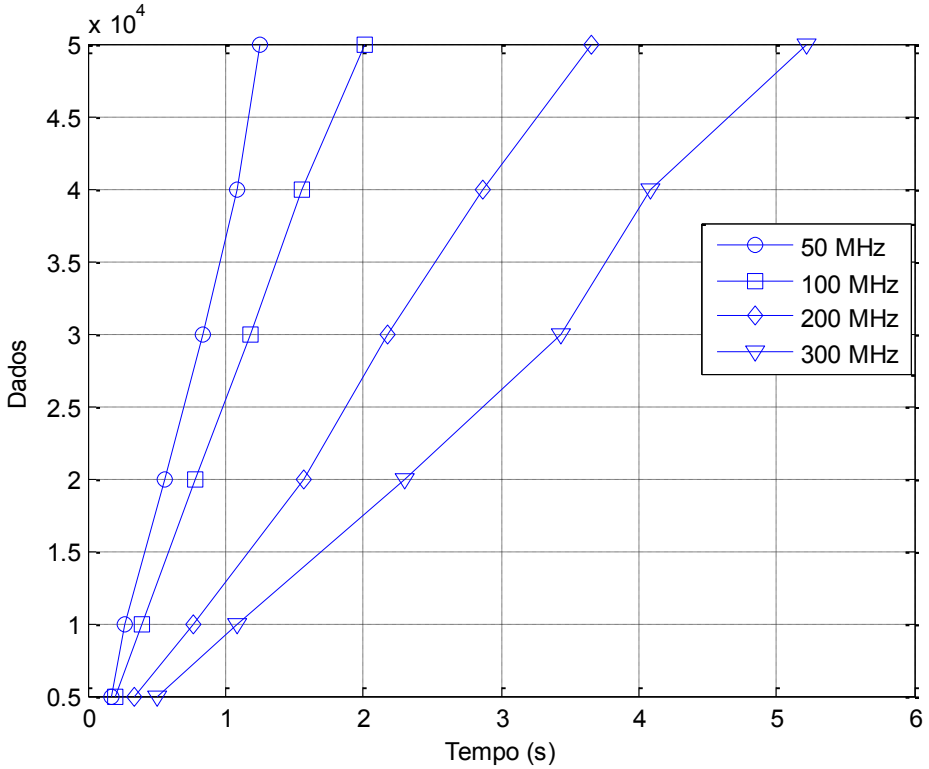

Figura 5.4 - Comparação temporal das frequências usadas

5.7 Comparação de tempos por dados

	FPGA	PC	*PowerPC* @ 50 MHz
Tempo por dado (s)	1,3E-8	2,8E-7	2,8E-5

Tabela 5.11 - Comparação temporal das diferentes plataformas

Comparando os tempos de cada plataforma podemos verificar que, sem dúvida, os melhores tempos são obtidos pela FPGA, sendo a plataforma com o melhor desempenho. A plataforma com o segundo melhor desempenho é o PC, apesar de ter um desempenho temporal uma ordem de grandeza pior que a FPGA. Os resultados do *PowerPC* são desapontantes, visto

serem extremamente piores do que o esperado, sendo esta plataforma a mais lenta das três. Como já foi referido, os motivos para isto não foram apurados, mas podem dever-se a vários factores, tais como o facto do método usado para obtenção de dados ser um método que altera o normal modo de funcionamento de um algoritmo.

5.8 Conclusões

A evolução da tecnologia relacionada com os sistemas computacionais sofreu uma enorme evolução ao longo dos anos, com o aparecimento de novas plataformas e com a evolução das mesmas. Ao longo desta dissertação foram referidas algumas, com especial destaque para os sistemas embutidos.

O estudo feito ao longo deste trabalho assentava na comparação de algumas plataformas diferentes, para tentar obter uma melhor perspectiva das vantagens e desvantagens dessas mesmas plataformas.

Olhando com atenção para este trabalho, verificamos que a parte teórica dos primeiros capítulos justifica em parte os resultados obtidos. Apesar de os resultados obtidos no PC e na FPGA serem completamente dentro do esperado, os resultados no *PowerPC* não podiam estar mais longe do esperado. Existem algumas razões para isto acontecer, no entanto não se conseguiram determinar quais.

5.9 Investigação Futura

Num estudo futuro, uma forte possibilidade de trabalho seria conseguir que os resultados do *PowerPC* se aproximassem do esperado, isto é conseguir reduzir o tempo de execução de cada algoritmo, assim como também conseguir com que o tempo de execução não aumente com o aumentar da frequência de relógio.

Seria também interessante usar algoritmos diferentes e mais exigentes em termos de recursos, de modo a saber se as diferenças temporais se alterariam. Como a tecnologia usada neste trabalho é relativamente antiga (Virtex4) seria também interessante repetir este estudo, mas usando tecnologias mais recentes.

Referências

1. Xilinx. “EDK Concepts, Tools, and Techniques – A Hands-On Guide to Effective Embedded System Design”, 2009
2. Xilinx. “EDK Profiling User Guide – A Guide to Profiling in EDK”
3. Digilent. “FX12 Board Reference Manual – Featuring the Xilinx Virtex-4 FPGA”, 2006
4. Xilinx. ”ISE Design Suite Software Manuals and Help - PDF Collection”,2009
5. Xilinx. “PowerPC Processor Reference Guide”, 2010
6. Xilinx, “Virtex-4 Family Overview”, 2010
7. http://www.xilinx.com/company/history.htm
8. http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC4
9. http://www.intel.com/pressroom/kits/events/moores_law_40th/index.htm
10. http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_project_navigator_overview.htm
11. http://wiki.altium.com/display/ADOH/XC4VLX25-10FF668C+-+Feature+Summary
12. Xilinx. “Revolutionary architecture for the next generation platform fpgas”, 2003.
13. Xilinx, “Virtex-4 FX12 PowerPC and MicroBlaze Virtex-4 FX12 PowerPC and MicroBlaze Edition Kit Reference Systems”, 2008
14. http://www.xilinx.com/company/gettingstarted/index.htm
15. Peter J. Ashenden. “Digital Design, An Embedded Systems Approach Using VHDL”, Morgan Kaufmann Publishers, 2008.
16. V. Sklyarov. “FPGA-based implementation of recursive algorithms”, Microprocessors and Microsystems. Special Issue on FPGAs: Applications and Designs, 2004, Vol 28/5-6 pp 197-211.
17. http://www.xilinx.com/company/about/index.htm#company-overview
18. D. Mihhailov, V. Sklyarov, I. Skliarova, A. Sudnitson, "Optimization of FPGA-based Circuits for Recursive Data Sorting", Proceedings of the 12th Biennial Baltic Electronics Conference - BEC 2010, Tallinn, Estonia, October 2010, pp. 129-132.
19. E. Cigan and N. Lall, “Integrating Matlab algorithms into FPGA design”, Xcell. Journal, vol. 53, pp. 37–39, 2005

20. http://www.wordiq.com/definition/Hardware_description_language#External_links
21. Rahul Dubey. “Introduction to Embedded System Design Using Field Programmable Gate Arrays”,Springer,2009
22. Digilent, “Digilent Nexys2 Board Reference Manual”, 2008

Anexos A – Xilinx Board Description FX12

```
ATTRIBUTE VENDOR = Nu Horizons
ATTRIBUTE SPEC_URL = www.nuhorizons.com
ATTRIBUTE NAME = Virtex-4 FX12 Evaluation Board
ATTRIBUTE REVISION = B1
ATTRIBUTE DESC = NuHorizons Virtex-4 Evaluation Board Rev B1
ATTRIBUTE LONG_DESC = 'This board utilizes XILINX FPGA XC4VFX12-10C. The board includes a
VGA video DAC, 16M X 32 SDRAM, 4M x 32 FLASH, 2X164 Character LCD I/F, RS232 serial port,
VGA port, 50MHz Clock Crystal, XCF08 Platform Flash for configuration, 8 LEDs, 4 push
button, programmable clock synthesizer, LTC Acquisition header and 3.3V, 2.5V, 1.8V and
1.2V supplies'

BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_CLOCK_V1
   ATTRIBUTE INSTANCE =sysclk
   PARAMETER CLK_FREQ =100000000, IO_IS=clk_freq, RANGE=(100000000) # 100 Mhz
   PORT SYSCLK = CLK_100MHZ_OSC, IO_IS=ext_clk
END
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_RESET_V1
   ATTRIBUTE INSTANCE = rst_0
   PARAMETER RST_POLARITY = 1, IO_IS=polarity, VALUE_NOTE=Active HIGH
   PORT FPGA.RESET = sys_rst, IO_IS=ext_rst
END
# RS232 1
# Uart interfaces with FPGA
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_UART_V1
  ATTRIBUTE INSTANCE = RS232_Uart
  PORT RX  = uart1_sin,  IO_IS=serial_in
  PORT TX  = uart1_sout, IO_IS=serial_out
END
# GPLED 8 LED's
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = LEDs_8Bit
  PARAMETER num_bits = 8, IO_IS=num_bits
  PARAMETER is_dual=0, IO_IS=is_dual
  PARAMETER bidir_data=1, IO_IS=is_bidir     # Bidir data pins
  PARAMETER all_inputs = 0, IO_IS=all_inputs
  PORT GPLED0 = GPLED_0, IO_IS = gpio_io[0]
  PORT GPLED1 = GPLED_1, IO_IS = gpio_io[1]
  PORT GPLED2 = GPLED_2, IO_IS = gpio_io[2]
  PORT GPLED3 = GPLED_3, IO_IS = gpio_io[3]
  PORT GPLED4 = GPLED_4, IO_IS = gpio_io[4]
  PORT GPLED5 = GPLED_5, IO_IS = gpio_io[5]
  PORT GPLED6 = GPLED_6, IO_IS = gpio_io[6]
  PORT GPLED7 = GPLED_7, IO_IS = gpio_io[7]
END
# Position push buttons
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = Push_Buttons_3
  PARAMETER num_bits = 3, IO_IS=num_bits
  PARAMETER is_dual=0, IO_IS=is_dual
  PARAMETER bidir_data=1, IO_IS=is_bidir     # Bidir data pins
  PARAMETER all_inputs = 1, IO_IS=all_inputs
  PORT PUSH_BUTTON0 = PUSH_BUTTON_0, IO_IS = gpio_io[0]
  PORT PUSH_BUTTON1 = PUSH_BUTTON_1, IO_IS = gpio_io[1]
  PORT PUSH_BUTTON2 = PUSH_BUTTON_2, IO_IS = gpio_io[2]
```

```
#  PORT PUSH_BUTTON3 = PUSH_BUTTON_3, IO_IS = gpio_io[3]
END

BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_MEMORY_V1
  ATTRIBUTE INSTANCE = DDR_SDRAM

  PARAMETER C_MEM_PARTNO = "MT46V16M16-5B", IO_IS = C_MEM_PARTNO
  PARAMETER C_BASEADDR = 0x00000000, IO_IS=C_BASEADDR, SHORT_DESC=DDR_SDRAM
  PARAMETER C_HIGHADDR = 0x03ffffff, IO_IS=C_HIGHADDR
  PARAMETER C_MEM_DATA_WIDTH = 32, IO_IS = C_MEM_DATA_WIDTH
  PARAMETER C_MEM_DQS_WIDTH = 4, IO_IS = C_MEM_DQS_WIDTH
  PARAMETER C_MEM_DM_WIDTH=4, IO_IS = C_MEM_DM_WIDTH
  PARAMETER C_MEM_TYPE=DDR, IO_IS = C_MEM_TYPE

  PARAMETER C_NUM_IDELAYCTRL = 3, IO_IS = C_NUM_IDELAYCTRL
  PARAMETER C_IDELAYCTRL_LOC = IDELAYCTRL_X1Y3-IDELAYCTRL_X0Y3-IDELAYCTRL_X0Y2, IO_IS =
C_IDELAYCTRL_LOC

  PORT A0  = ddr_addr_12_,  IO_IS = ddr_address[12]
  PORT A1  = ddr_addr_11_,  IO_IS = ddr_address[11]
  PORT A2  = ddr_addr_10_,  IO_IS = ddr_address[10]
  PORT A3  = ddr_addr_9_,  IO_IS = ddr_address[9]
  PORT A4  = ddr_addr_8_,  IO_IS = ddr_address[8]
  PORT A5  = ddr_addr_7_,  IO_IS = ddr_address[7]
  PORT A6  = ddr_addr_6_,  IO_IS = ddr_address[6]
  PORT A7  = ddr_addr_5_,  IO_IS = ddr_address[5]
  PORT A8  = ddr_addr_4_,  IO_IS = ddr_address[4]
  PORT A9  = ddr_addr_3_,  IO_IS = ddr_address[3]
  PORT A10 = ddr_addr_2_, IO_IS = ddr_address[2]
  PORT A11 = ddr_addr_1_, IO_IS = ddr_address[1]
  PORT A12 = ddr_addr_0_, IO_IS = ddr_address[0]
  PORT BA0 = ddr_ba_1_,    IO_IS = ddr_bankaddr[1]
  PORT BA1 = ddr_ba_0_,    IO_IS = ddr_bankaddr[0]
  PORT CAS = ddr_cas_n,    IO_IS = ddr_col_addr_select
  PORT CKE = ddr_cke,      IO_IS = ddr_clock_enable
  PORT CS  = ddr_cs_n,     IO_IS = ddr_chip_select
  PORT RAS = ddr_ras_n,    IO_IS = ddr_row_addr_select
  PORT WE  = ddr_we_n,     IO_IS = ddr_write_enable
  # Point-to-point connections for data pins from FPGA to DDR SDRAM devices
  PORT U6.DM = ddr_dm_0_,  IO_IS = ddr_data_mask[0]
  PORT U7.DM = ddr_dm_1_,  IO_IS = ddr_data_mask[1]
  PORT U8.DM = ddr_dm_2_,  IO_IS = ddr_data_mask[2]
  PORT U9.DM = ddr_dm_3_,  IO_IS = ddr_data_mask[3]
  PORT U6.DQS = ddr_dqs_0_, IO_IS = ddr_data_strobe[0]
  PORT U7.DQS = ddr_dqs_1_, IO_IS = ddr_data_strobe[1]
  PORT U8.DQS = ddr_dqs_2_, IO_IS = ddr_data_strobe[2]
  PORT U9.DQS = ddr_dqs_3_, IO_IS = ddr_data_strobe[3]
  PORT U6.DQ0 = ddr_dq_0_,  IO_IS = ddr_data[0]
  PORT U6.DQ1 = ddr_dq_1_,  IO_IS = ddr_data[1]
  PORT U6.DQ2 = ddr_dq_2_,  IO_IS = ddr_data[2]
  PORT U6.DQ3 = ddr_dq_3_,  IO_IS = ddr_data[3]
  PORT U6.DQ4 = ddr_dq_4_,  IO_IS = ddr_data[4]
  PORT U6.DQ5 = ddr_dq_5_,  IO_IS = ddr_data[5]
  PORT U6.DQ6 = ddr_dq_6_,  IO_IS = ddr_data[6]
  PORT U6.DQ7 = ddr_dq_7_,  IO_IS = ddr_data[7]
  PORT U7.DQ0 = ddr_dq_8_,  IO_IS = ddr_data[8]
  PORT U7.DQ1 = ddr_dq_9_,  IO_IS = ddr_data[9]
  PORT U7.DQ2 = ddr_dq_10_, IO_IS = ddr_data[10]
  PORT U7.DQ3 = ddr_dq_11_, IO_IS = ddr_data[11]
  PORT U7.DQ4 = ddr_dq_12_, IO_IS = ddr_data[12]
  PORT U7.DQ5 = ddr_dq_13_, IO_IS = ddr_data[13]
  PORT U7.DQ6 = ddr_dq_14_, IO_IS = ddr_data[14]
  PORT U7.DQ7 = ddr_dq_15_, IO_IS = ddr_data[15]
  PORT U8.DQ0 = ddr_dq_16_, IO_IS = ddr_data[16]
  PORT U8.DQ1 = ddr_dq_17_, IO_IS = ddr_data[17]
  PORT U8.DQ2 = ddr_dq_18_, IO_IS = ddr_data[18]
  PORT U8.DQ3 = ddr_dq_19_, IO_IS = ddr_data[19]
  PORT U8.DQ4 = ddr_dq_20_, IO_IS = ddr_data[20]
  PORT U8.DQ5 = ddr_dq_21_, IO_IS = ddr_data[21]
  PORT U8.DQ6 = ddr_dq_22_, IO_IS = ddr_data[22]
  PORT U8.DQ7 = ddr_dq_23_, IO_IS = ddr_data[23]
  PORT U9.DQ0 = ddr_dq_24_, IO_IS = ddr_data[24]
  PORT U9.DQ1 = ddr_dq_25_, IO_IS = ddr_data[25]
```

```
  PORT U9.DQ2 = ddr_dq_26_, IO_IS = ddr_data[26]
  PORT U9.DQ3 = ddr_dq_27_, IO_IS = ddr_data[27]
  PORT U9.DQ4 = ddr_dq_28_, IO_IS = ddr_data[28]
  PORT U9.DQ5 = ddr_dq_29_, IO_IS = ddr_data[29]
  PORT U9.DQ6 = ddr_dq_30_, IO_IS = ddr_data[30]
  PORT U9.DQ7 = ddr_dq_31_, IO_IS = ddr_data[31]

  PORT DDR_FPGA_CK   = ddr_clk, IO_IS=ddr_clk
  PORT DDR_Clkn = ddr_clk_n, IO_IS = ddr_clk_n
 #  PORT DDR_FPGA_CK_N = ddr_clk_n, IO_IS=ddr_clk_out_n
 #  PORT DDR_FB_CLK    = ddr_clk_fb, IO_IS=feedback_clock,FEEDBACK_PHASE=12
 # PORT DDR_FB_CLK_OUT    = ddr_clk_fb_out, IO_IS=feedback_clock_out
END

# Ethernet MAC
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_TEMAC_V1
  ATTRIBUTE INSTANCE = Ethernet_MAC
  ATTRIBUTE EXCLUSIVE =  Ethernet
  # hard temac params
  PARAMETER C_PHY_TYPE = 1, IO_IS=C_PHY_TYPE
  PARAMETER C_EMAC1_PRESENT = 0, IO_IS=C_EMAC1_PRESENT
  # plb_temac params
  PARAMETER C_TEMAC_INST = 0, IO_IS=C_TEMAC_INST
  PARAMETER C_TEMAC_BOTH_USED = 0, IO_IS=C_TEMAC_BOTH_USED
  PARAMETER C_MAC_FIFO_DEPTH = 64, IO_IS=C_MAC_FIFO_DEPTH

  PORT RESET    = phy_rst_n, IO_IS = TemacPhy_RST_n, INITIALVAL = VCC
  PORT MDINT    = phy_mii_int_n, IO_IS = Interrupt, SIGIS=INTERRUPT, SENSITIVITY=LEVEL_LOW,
INTERRUPT_PRIORITY=MEDIUM
  PORT TXD3     = phy_tx_data_3, IO_IS = MII_TXD_0[3]
  PORT TXD2     = phy_tx_data_2, IO_IS = MII_TXD_0[2]
  PORT TXD1     = phy_tx_data_1, IO_IS = MII_TXD_0[1]
  PORT TXD0     = phy_tx_data_0, IO_IS = MII_TXD_0[0]
  PORT TX_EN    = phy_tx_en,     IO_IS = MII_TX_EN_0
  PORT TX_CLK   = phy_tx_clk,    IO_IS = MII_TX_CLK_0
  PORT RX_ER    = phy_rx_er,     IO_IS = MII_RX_ER_0
  PORT RX_CLK   = phy_rx_clk,    IO_IS = MII_RX_CLK_0
  PORT RX_DV    = phy_dv,        IO_IS = MII_RX_DV_0
  PORT RXD0     = phy_rx_data_0, IO_IS = MII_RXD_0[0]
  PORT RXD1     = phy_rx_data_1, IO_IS = MII_RXD_0[1]
  PORT RXD2     = phy_rx_data_2, IO_IS = MII_RXD_0[2]
  PORT RXD3     = phy_rx_data_3, IO_IS = MII_RXD_0[3]
  PORT PHY_MDC  = phy_mii_clk,   IO_IS = MDC_0
  PORT PHY_MDIO = phy_mii_data,  IO_IS = MDIO_0
  PORT CRS      = phy_crs,       IO_IS=ETH_CRS
  PORT COL      = phy_col,       IO_IS=ETH_COL
END
BEGIN IO_INTERFACE
   ATTRIBUTE IOTYPE = XIL_EMC_V1
   ATTRIBUTE INSTANCE = FLASH_4Mx32
   ATTRIBUTE EXCLUSIVE = FLASH
   PARAMETER C_NUM_BANKS_MEM =     1, IO_IS=C_NUM_BANKS_MEM
   PARAMETER C_DEV_MIR_ENABLE =    0, IO_IS=C_DEV_MIR_ENABLE
   PARAMETER C_MAX_MEM_WIDTH =     32, IO_IS=C_MAX_MEM_WIDTH
#PARAMETER C_INCLUDE_BURST_CACHELN_SUPPORT = 1, IO_IS=C_INCLUDE_BURST_CACHELN_SUPPORT
   PARAMETER C_INCLUDE_DATAWIDTH_MATCHING_0 =  1, IO_IS=C_INCLUDE_DATAWIDTH_MATCHING_0
   PARAMETER C_MEM0_BASEADDR =    0x0, IO_IS=C_MEM0_BASEADDR, SHORT_DESC=FLASH_2Mx32,
MEMORY_TYPE=FLASH
   PARAMETER C_MEM0_HIGHADDR =    0xFFFFFF, IO_IS=C_MEM0_HIGHADDR
   PARAMETER C_MEM0_WIDTH =     32, IO_IS=C_MEM0_WIDTH
   PARAMETER C_SYNCH_MEM_0 =     0, IO_IS=C_SYNCH_MEM_0
   PARAMETER C_SYNCH_PIPEDELAY_0 =    2, IO_IS=C_SYNCH_PIPEDELAY_0
   PARAMETER C_READ_ADDR_TO_OUT_SLOW_PS_0 = 110000,  IO_IS = C_READ_ADDR_TO_OUT_SLOW_PS_0
   PARAMETER C_WRITE_ADDR_TO_OUT_SLOW_PS_0 = 55000, IO_IS = C_WRITE_ADDR_TO_OUT_SLOW_PS_0
   PARAMETER C_WRITE_MIN_PULSE_WIDTH_PS_0 =   70000,  IO_IS = C_WRITE_MIN_PULSE_WIDTH_PS_0
   PARAMETER C_READ_ADDR_TO_OUT_FAST_PS_0 =   150000,  IO_IS = C_READ_ADDR_TO_OUT_FAST_PS_0
   PARAMETER C_WRITE_ADDR_TO_OUT_FAST_PS_0 =   55000, IO_IS = C_WRITE_ADDR_TO_OUT_FAST_PS_0
   PARAMETER C_READ_RECOVERY_BEFORE_WRITE_PS_0 =  15000, IO_IS =
C READ RECOVERY BEFORE WRITE PS 0
   PARAMETER C_WRITE_RECOVERY_BEFORE_READ_PS_0 =  35000, IO_IS =
C WRITE RECOVERY BEFORE READ PS 0
   PORT A0 = CONN_FLASH_A0, IO_IS =    emc_addr[29]
   PORT A1 = CONN_FLASH_A1, IO_IS =    emc_addr[28]
```

```
    PORT A2 = CONN_FLASH_A2, IO_IS =    emc_addr[27]
    PORT A3 = CONN_FLASH_A3, IO_IS =    emc_addr[26]
    PORT A4 = CONN_FLASH_A4, IO_IS =    emc_addr[25]
    PORT A5 = CONN_FLASH_A5, IO_IS =    emc_addr[24]
    PORT A6 = CONN_FLASH_A6, IO_IS =    emc_addr[23]
    PORT A7 = CONN_FLASH_A7, IO_IS =    emc_addr[22]
    PORT A8 = CONN_FLASH_A8, IO_IS =    emc_addr[21]
    PORT A9 = CONN_FLASH_A9, IO_IS =    emc_addr[20]
    PORT A10 = CONN_FLASH_A10, IO_IS =    emc_addr[19]
    PORT A11 = CONN_FLASH_A11, IO_IS =    emc_addr[18]
    PORT A12 = CONN_FLASH_A12, IO_IS =    emc_addr[17]
    PORT A13 = CONN_FLASH_A13, IO_IS =    emc_addr[16]
    PORT A14 = CONN_FLASH_A14, IO_IS =    emc_addr[15]
    PORT A15 = CONN_FLASH_A15, IO_IS =    emc_addr[14]
    PORT A16 = CONN_FLASH_A16, IO_IS =    emc_addr[13]
    PORT A17 = CONN_FLASH_A17, IO_IS =    emc_addr[12]
    PORT A18 = CONN_FLASH_A18, IO_IS =    emc_addr[11]
    PORT A19 = CONN_FLASH_A19, IO_IS =    emc_addr[10]
    PORT A20 = CONN_FLASH_A20, IO_IS =    emc_addr[9]
    PORT A21 = CONN_FLASH_A21, IO_IS =    emc_addr[8]
    PORT WEN = CONN_FLASH_WEN, IO_IS=emc_wen
    PORT D0 = CONN_FLASH_D0, IO_IS = emc_data[31]
    PORT D1 = CONN_FLASH_D1, IO_IS = emc_data[30]
    PORT D2 = CONN_FLASH_D2, IO_IS = emc_data[29]
    PORT D3 = CONN_FLASH_D3, IO_IS = emc_data[28]
    PORT D4 = CONN_FLASH_D4, IO_IS = emc_data[27]
    PORT D5 = CONN_FLASH_D5, IO_IS = emc_data[26]
    PORT D6 = CONN_FLASH_D6,  IO_IS = emc_data[25]
    PORT D7 = CONN_FLASH_D7, IO_IS = emc_data[24]
    PORT D8 = CONN_FLASH_D8, IO_IS = emc_data[23]
    PORT D9 = CONN_FLASH_D9, IO_IS = emc_data[22]
    PORT D10 = CONN_FLASH_D10, IO_IS = emc_data[21]
    PORT D11 = CONN_FLASH_D11, IO_IS = emc_data[20]
    PORT D12 = CONN_FLASH_D12, IO_IS = emc_data[19]
    PORT D13 = CONN_FLASH_D13, IO_IS = emc_data[18]
    PORT D14 = CONN_FLASH_D14, IO_IS = emc_data[17]
    PORT D15 = CONN_FLASH_D15, IO_IS = emc_data[16]
    PORT D16 = CONN_FLASH_D16, IO_IS = emc_data[15]
    PORT D17 = CONN_FLASH_D17, IO_IS = emc_data[14]
    PORT D18 = CONN_FLASH_D18, IO_IS = emc_data[13]
    PORT D19 = CONN_FLASH_D19, IO_IS = emc_data[12]
    PORT D20 = CONN_FLASH_D20, IO_IS = emc_data[11]
    PORT D21 = CONN_FLASH_D21, IO_IS = emc_data[10]
    PORT D22 = CONN_FLASH_D22, IO_IS = emc_data[9]
    PORT D23 = CONN_FLASH_D23, IO_IS = emc_data[8]
    PORT D24 = CONN_FLASH_D24, IO_IS = emc_data[7]
    PORT D25 = CONN_FLASH_D25, IO_IS = emc_data[6]
    PORT D26 = CONN_FLASH_D26, IO_IS = emc_data[5]
    PORT D27 = CONN_FLASH_D27, IO_IS = emc_data[4]
    PORT D28 = CONN_FLASH_D28, IO_IS = emc_data[3]
    PORT D29 = CONN_FLASH_D29, IO_IS = emc_data[2]
    PORT D30 = CONN_FLASH_D30, IO_IS = emc_data[1]
    PORT D31 = CONN_FLASH_D31, IO_IS = emc_data[0]
    PORT OEN = CONN_FLASH_OEN, IO_IS = emc_oen[0]
    PORT CE  = CONN_FLASH_CE,  IO_IS = emc_ce[0]
END
BEGIN FPGA
  ATTRIBUTE INSTANCE = fpga_0
  ATTRIBUTE FAMILY = virtex4
  ATTRIBUTE DEVICE = xc4vfx12
  ATTRIBUTE PACKAGE = sf363
  ATTRIBUTE SPEED_GRADE = -10
  ATTRIBUTE JTAG_POSITION = 1
  PORT CLK_100MHZ_OSC  = CLK_100MHZ_OSC,  UCF_NET_STRING=("LOC=Y5", "IOSTANDARD =
LVCMOS33") # Input CLK
# RS232 1
  PORT RXD = uart1_sin, UCF_NET_STRING=("LOC=R17", "IOSTANDARD = LVCMOS33")
  PORT TXD = uart1_sout, UCF_NET_STRING=("LOC=R18, "IOSTANDARD = LVCMOS33")
# GPLED
  PORT GPLED0 = GPLED_0, UCF_NET_STRING=("LOC=U2", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED1 = GPLED_1, UCF_NET_STRING=("LOC=U3", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
```

```
  PORT GPLED2 = GPLED_2, UCF_NET_STRING=("LOC=T3", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED3 = GPLED_3, UCF_NET_STRING=("LOC=T4", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED4 = GPLED_4, UCF_NET_STRING=("LOC=R5", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED5 = GPLED_5, UCF_NET_STRING=("LOC=R6", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED6 = GPLED_6, UCF_NET_STRING=("LOC=R3", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
  PORT GPLED7 = GPLED_7, UCF_NET_STRING=("LOC=R4", "IOSTANDARD = LVCMOS33", "PULLUP", "SLEW
= SLOW", "DRIVE = 2", "TIG")
# push buttons
  PORT sys_rst_pin = sys_rst, UCF_NET_STRING=("LOC=N2", "IOSTANDARD = LVCMOS33", "TIG")
  PORT PUSH_BUTTON0 = PUSH_BUTTON_0, UCF_NET_STRING=("LOC=P2", "IOSTANDARD = LVCMOS33",
"TIG")
  PORT PUSH_BUTTON1 = PUSH_BUTTON_1, UCF_NET_STRING=("LOC=P5", "IOSTANDARD = LVCMOS33",
"TIG")
  PORT PUSH_BUTTON2 = PUSH_BUTTON_2, UCF_NET_STRING=("LOC=P4", "IOSTANDARD = LVCMOS33",
"TIG")

# DDR SDRAM 64Mx32
  PORT DDR_CLK    = ddr_clk,     UCF_NET_STRING=("LOC=P16", "IOSTANDARD = DIFF_SSTL2_II")
  PORT DDR_CLK_FB = ddr_clk_fb,  UCF_NET_STRING=("LOC=B7",  "IOSTANDARD = LVCMOS25")
  PORT DDR_CLK_N  = ddr_clk_n,   UCF_NET_STRING=("LOC=P17", "IOSTANDARD = DIFF_SSTL2_II")
  PORT DDR_A00    = ddr_addr_0_, UCF_NET_STRING=("LOC=E16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A01    = ddr_addr_1_, UCF_NET_STRING=("LOC=D16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A02    = ddr_addr_2_, UCF_NET_STRING=("LOC=C16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A03    = ddr_addr_3_, UCF_NET_STRING=("LOC=C15", "IOSTANDARD = SSTL2_I")
  PORT DDR_A04    = ddr_addr_4_, UCF_NET_STRING=("LOC=G16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A05    = ddr_addr_5_, UCF_NET_STRING=("LOC=H16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A06    = ddr_addr_6_, UCF_NET_STRING=("LOC=J16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A07    = ddr_addr_7_, UCF_NET_STRING=("LOC=J15", "IOSTANDARD = SSTL2_I")
  PORT DDR_A08    = ddr_addr_8_, UCF_NET_STRING=("LOC=K16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A09    = ddr_addr_9_, UCF_NET_STRING=("LOC=K17", "IOSTANDARD = SSTL2_I")
  PORT DDR_A10    = ddr_addr_10_, UCF_NET_STRING=("LOC=F16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A11    = ddr_addr_11_, UCF_NET_STRING=("LOC=M16", "IOSTANDARD = SSTL2_I")
  PORT DDR_A12    = ddr_addr_12_, UCF_NET_STRING=("LOC=M15", "IOSTANDARD = SSTL2_I")
  PORT DDR_BA0    = ddr_ba_0_,   UCF_NET_STRING=("LOC=D17", "IOSTANDARD = SSTL2_I")
  PORT DDR_BA1    = ddr_ba_1_,   UCF_NET_STRING=("LOC=C18", "IOSTANDARD = SSTL2_I")
  PORT DDR_CAS_N  = ddr_cas_n,   UCF_NET_STRING=("LOC=F17", "IOSTANDARD = SSTL2_I")
  PORT DDR_CKE    = ddr_cke,     UCF_NET_STRING=("LOC=N16", "IOSTANDARD = SSTL2_I")
  PORT DDR_CS_N   = ddr_cs_n,    UCF_NET_STRING=("LOC=D18", "IOSTANDARD = SSTL2_I")
  PORT DDR_RAS_N  = ddr_ras_n,   UCF_NET_STRING=("LOC=E18", "IOSTANDARD = SSTL2_I")
  PORT DDR_WE_N   = ddr_we_n,    UCF_NET_STRING=("LOC=F18", "IOSTANDARD = SSTL2_I")
  PORT DDR_DQM0 = ddr_dm_0_,  UCF_NET_STRING=("LOC=A13", "IOSTANDARD = SSTL2_I")
  PORT DDR_DQM1 = ddr_dm_1_,  UCF_NET_STRING=("LOC=B13", "IOSTANDARD = SSTL2_I")
  PORT DDR_DQM2 = ddr_dm_2_,  UCF_NET_STRING=("LOC=A14", "IOSTANDARD = SSTL2_I")
  PORT DDR_DQM3 = ddr_dm_3_,  UCF_NET_STRING=("LOC=B14", "IOSTANDARD = SSTL2_I")
  PORT DDR_DQS0 = ddr_dqs_0_, UCF_NET_STRING=("LOC=J19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQS1 = ddr_dqs_1_, UCF_NET_STRING=("LOC=N18", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQS2 = ddr_dqs_2_, UCF_NET_STRING=("LOC=A15", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQS3 = ddr_dqs_3_, UCF_NET_STRING=("LOC=B15", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ00 = ddr_dq_0_,  UCF_NET_STRING=("LOC=P20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ01 = ddr_dq_1_,  UCF_NET_STRING=("LOC=P19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ02 = ddr_dq_2_,  UCF_NET_STRING=("LOC=N19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ03 = ddr_dq_3_,  UCF_NET_STRING=("LOC=M20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ04 = ddr_dq_4_,  UCF_NET_STRING=("LOC=M19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ05 = ddr_dq_5_,  UCF_NET_STRING=("LOC=L20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ06 = ddr_dq_6_,  UCF_NET_STRING=("LOC=K19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ07 = ddr_dq_7_,  UCF_NET_STRING=("LOC=K20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ08 = ddr_dq_8_,  UCF_NET_STRING=("LOC=N17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ09 = ddr_dq_9_,  UCF_NET_STRING=("LOC=M17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ10 = ddr_dq_10_, UCF_NET_STRING=("LOC=M18", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ11 = ddr_dq_11_, UCF_NET_STRING=("LOC=K18", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ12 = ddr_dq_12_, UCF_NET_STRING=("LOC=J18", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ13 = ddr_dq_13_, UCF_NET_STRING=("LOC=J17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ14 = ddr_dq_14_, UCF_NET_STRING=("LOC=H17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ15 = ddr_dq_15_, UCF_NET_STRING=("LOC=G17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ16 = ddr_dq_16_, UCF_NET_STRING=("LOC=C20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ17 = ddr_dq_17_, UCF_NET_STRING=("LOC=C19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ18 = ddr_dq_18_, UCF_NET_STRING=("LOC=B19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ19 = ddr_dq_19_, UCF_NET_STRING=("LOC=B18", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ20 = ddr_dq_20_, UCF_NET_STRING=("LOC=A18", "IOSTANDARD = SSTL2_II")
```

```
  PORT DDR_DQ21 = ddr_dq_21_, UCF_NET_STRING=("LOC=B17", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ22 = ddr_dq_22_, UCF_NET_STRING=("LOC=B16", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ23 = ddr_dq_23_, UCF_NET_STRING=("LOC=A16", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ24 = ddr_dq_24_, UCF_NET_STRING=("LOC=D19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ25 = ddr_dq_25_, UCF_NET_STRING=("LOC=E20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ26 = ddr_dq_26_, UCF_NET_STRING=("LOC=F19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ27 = ddr_dq_27_, UCF_NET_STRING=("LOC=F20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ28 = ddr_dq_28_, UCF_NET_STRING=("LOC=G19", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ29 = ddr_dq_29_, UCF_NET_STRING=("LOC=G20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ30 = ddr_dq_30_, UCF_NET_STRING=("LOC=H20", "IOSTANDARD = SSTL2_II")
  PORT DDR_DQ31 = ddr_dq_31_, UCF_NET_STRING=("LOC=H18", "IOSTANDARD = SSTL2_II")
  # 10/100 Ethernet MAC
  PORT PHY_RESET  = phy_rst_n,     UCF_NET_STRING=("LOC=F15", "IOSTANDARD = LVCMOS25",
"TIG")
  PORT PHY_CRS    = phy_crs,       UCF_NET_STRING=("LOC=A7", "IOSTANDARD = LVCMOS25")
  PORT PHY_COL    = phy_col,       UCF_NET_STRING=("LOC=E7", "IOSTANDARD = LVCMOS25")
  PORT PHY_TXD3   = phy_tx_data_3, UCF_NET_STRING=("LOC=C9", "IOSTANDARD = LVCMOS25")
  PORT PHY_TXD2   = phy_tx_data_2, UCF_NET_STRING=("LOC=D9", "IOSTANDARD = LVCMOS25")
  PORT PHY_TXD1   = phy_tx_data_1, UCF_NET_STRING=("LOC=C8", "IOSTANDARD = LVCMOS25")
  PORT PHY_TXD0   = phy_tx_data_0, UCF_NET_STRING=("LOC=D8", "IOSTANDARD = LVCMOS25")
  PORT PHY_TX_EN  = phy_tx_en,     UCF_NET_STRING=("LOC=E14", "IOSTANDARD = LVCMOS25")
  PORT PHY_TX_CLK = phy_tx_clk,    UCF_NET_STRING=("LOC=B12", "IOSTANDARD = LVCMOS25")
#  PORT PHY_TX_ER  = phy_tx_er,     UCF_NET_STRING=("LOC=E7", "IOSTANDARD = LVCMOS25")
  PORT PHY_RX_ER  = phy_rx_er,     UCF_NET_STRING=("LOC=D6", "IOSTANDARD = LVCMOS25")
  PORT PHY_RX_CLK = phy_rx_clk,    UCF_NET_STRING=("LOC=C11", "IOSTANDARD = LVCMOS25")
  PORT PHY_RX_DV  = phy_dv,        UCF_NET_STRING=("LOC=E6", "IOSTANDARD = LVCMOS25")
  PORT PHY_RXD0   = phy_rx_data_0, UCF_NET_STRING=("LOC=C10", "IOSTANDARD = LVCMOS25")
  PORT PHY_RXD1   = phy_rx_data_1, UCF_NET_STRING=("LOC=A11", "IOSTANDARD = LVCMOS25")
  PORT PHY_RXD2   = phy_rx_data_2, UCF_NET_STRING=("LOC=B11", "IOSTANDARD = LVCMOS25")
  PORT PHY_RXD3   = phy_rx_data_3, UCF_NET_STRING=("LOC=A10", "IOSTANDARD = LVCMOS25")
  PORT PHY_MDC    = phy_mii_clk,   UCF_NET_STRING=("LOC=D15", "IOSTANDARD = LVCMOS25")
  PORT PHY_MDIO   = phy_mii_data,  UCF_NET_STRING=("LOC=F6", "IOSTANDARD = LVCMOS25")
# FLASH
   PORT FLASH_A0 = CONN_FLASH_A0, UCF_NET_STRING=("LOC=M6", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A1 = CONN_FLASH_A1, UCF_NET_STRING=("LOC=M5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A2 = CONN_FLASH_A2, UCF_NET_STRING=("LOC=L5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A3 = CONN_FLASH_A3, UCF_NET_STRING=("LOC=M4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A4 = CONN_FLASH_A4, UCF_NET_STRING=("LOC=M3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A5 = CONN_FLASH_A5, UCF_NET_STRING=("LOC=L4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A6 = CONN_FLASH_A6, UCF_NET_STRING=("LOC=K3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A7 = CONN_FLASH_A7, UCF_NET_STRING=("LOC=K4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A8 = CONN_FLASH_A8, UCF_NET_STRING=("LOC=F4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A9 = CONN_FLASH_A9, UCF_NET_STRING=("LOC=E3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A10 = CONN_FLASH_A10, UCF_NET_STRING=("LOC=D3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A11 = CONN_FLASH_A11, UCF_NET_STRING=("LOC=C3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A12 = CONN_FLASH_A12, UCF_NET_STRING=("LOC=D4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A13 = CONN_FLASH_A13, UCF_NET_STRING=("LOC=C4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A14 = CONN_FLASH_A14, UCF_NET_STRING=("LOC=D5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A15 = CONN_FLASH_A15, UCF_NET_STRING=("LOC=C5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A16 = CONN_FLASH_A16, UCF_NET_STRING=("LOC=C6", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A17 = CONN_FLASH_A17, UCF_NET_STRING=("LOC=J3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A18 = CONN_FLASH_A18, UCF_NET_STRING=("LOC=J4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A19 = CONN_FLASH_A19, UCF_NET_STRING=("LOC=F3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A20 = CONN_FLASH_A20, UCF_NET_STRING=("LOC=G4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_A21 = CONN_FLASH_A21, UCF_NET_STRING=("LOC=H3", "IOSTANDARD = LVCMOS33")
   PORT FLASH_WEN = CONN_FLASH_WEN, UCF_NET_STRING=("LOC=H4", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D0 = CONN_FLASH_D0, UCF_NET_STRING=("LOC=K5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D1 = CONN_FLASH_D1, UCF_NET_STRING=("LOC=J6", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D2 = CONN_FLASH_D2, UCF_NET_STRING=("LOC=G5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D3 = CONN_FLASH_D3, UCF_NET_STRING=("LOC=E5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D4 = CONN_FLASH_D4, UCF_NET_STRING=("LOC=M1", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D5 = CONN_FLASH_D5, UCF_NET_STRING=("LOC=L2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D6 = CONN_FLASH_D6, UCF_NET_STRING=("LOC=K2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D7 = CONN_FLASH_D7, UCF_NET_STRING=("LOC=H1", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D8 = CONN_FLASH_D8, UCF_NET_STRING=("LOC=J5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D9 = CONN_FLASH_D9, UCF_NET_STRING=("LOC=H5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D10 = CONN_FLASH_D10, UCF_NET_STRING=("LOC=F5", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D11 = CONN_FLASH_D11, UCF_NET_STRING=("LOC=M2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D12 = CONN_FLASH_D12, UCF_NET_STRING=("LOC=L1", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D13 = CONN_FLASH_D13, UCF_NET_STRING=("LOC=K1", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D14 = CONN_FLASH_D14, UCF_NET_STRING=("LOC=J2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D15 = CONN_FLASH_D15, UCF_NET_STRING=("LOC=H2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D16 = CONN_FLASH_D16, UCF_NET_STRING=("LOC=G2", "IOSTANDARD = LVCMOS33")
   PORT FLASH_D17 = CONN_FLASH_D17, UCF_NET_STRING=("LOC=F2", "IOSTANDARD = LVCMOS33")
```

```
    PORT FLASH_D18 = CONN_FLASH_D18, UCF_NET_STRING=("LOC=E2", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D19 = CONN_FLASH_D19, UCF_NET_STRING=("LOC=C1", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D20 = CONN_FLASH_D20, UCF_NET_STRING=("LOC=B2", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D21 = CONN_FLASH_D21, UCF_NET_STRING=("LOC=A3", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D22 = CONN_FLASH_D22, UCF_NET_STRING=("LOC=B5", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D23 = CONN_FLASH_D23, UCF_NET_STRING=("LOC=B6", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D24 = CONN_FLASH_D24, UCF_NET_STRING=("LOC=F1", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D25 = CONN_FLASH_D25, UCF_NET_STRING=("LOC=E1", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D26 = CONN_FLASH_D26, UCF_NET_STRING=("LOC=D2", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D27 = CONN_FLASH_D27, UCF_NET_STRING=("LOC=C2", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D28 = CONN_FLASH_D28, UCF_NET_STRING=("LOC=B3", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D29 = CONN_FLASH_D29, UCF_NET_STRING=("LOC=B4", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D30 = CONN_FLASH_D30, UCF_NET_STRING=("LOC=A5", "IOSTANDARD = LVCMOS33")
    PORT FLASH_D31 = CONN_FLASH_D31, UCF_NET_STRING=("LOC=A6", "IOSTANDARD = LVCMOS33")
    PORT FLASH_OEN = CONN_FLASH_OEN, UCF_NET_STRING=("LOC=G1", "IOSTANDARD = LVCMOS33")

END
```

Anexo B – Código utilizado no PC e no PowerPC

CTree.h

```
#include "iostream.h"

template <class T> struct treenode
    {   T val ; // valor dum objecto do tipo T
    int count; // Número de itens com o valor val
    treenode <T> *lnode; // Ponteiro para o nó esquerdo
    treenode <T> *rnode; // Ponteiro para o nó direito
    };
class CTree
{
public:
    CTree(void);
    ~CTree(void);
    };


template< class T>
treenode<T> *addnode(treenode<T> *node, T value)
{
    if (node==NULL) // construir um novo nó do tipo treenode<T>
    {
        if ((node = new treenode<T>) == NULL)
            cerr << "memory allocation error\n"; // excepção
        node->val = value;  // novo valor do nó
        node->count =1; // So um nó
        node->rnode = node->lnode = NULL;
    }
    else if(value==node->val)
        node->count++;  // Incrementar o contador
    else if(value < node->val)
        // examina a árvore ou constrói o nó da esquerda
        node->lnode = addnode(node->lnode,value);
    else
        //examina a árvore ou constrói o nó da direita
        node->rnode = addnode(node->rnode,value);
    return node; //devolve o ponteiro ao nó actual
}


//Ordena e imprimi valores
template<class T> void treesort(treenode<T> *node)
{
    if(node!=NULL) // se existir nó
    {
        treesort(node->lnode); //ordenar nó da árvore da esquerda
        //Mostrar valor após retorno hierárquico
        cout << "value - " << node->val << "; repeated - " << node->count << endl;
        treesort(node->rnode); // Ordenar árvore da direita
    }
}
```

Main.cpp

```
#include <iostream>
#include "CTree.h"
using namespace std;


int gerador()
{

    // semente inicial = 5323
    static unsigned int nSeed = 5323;

    // Gerar um valor novo a partir da semente inicial
    // Devido ao uso de grandes constants e overflow é difícil dizer qual
    // será o número gerado
    nSeed = (8253729 * nSeed + 2396403);


    // Usa a semente e elimina os valores superiores a 10000
    return nSeed  % 10000;
}


int main (void)
{
    int i;
    // declara o endereço do nó inicial de uma árvore para armazenar             //inteiros
    treenode<int> *root =NULL;
    //Escolhe números aleatórios até 1000 números
    for (i=1;i<=1000;i++)
        root = addnode(root, gerador()); // adiciona valor a árvore
    treesort(root); // ordenar os dados
    return 0;
}
```